PUBLIC
SQL Anywhere Server
Document Version: 17.01.0 – 2021-10-15

SQL Anywhere - SQL Reference

THE BEST RUN SAP

Content

1 SQL Anywhere Server - SQL Reference

This book describes the system procedures and the catalog (system tables and views) included in SQL Anywhere. It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).

In this section:

1.1 SQL Language Elements

There are several SQL language elements you can use.

In this section:

Operators [page 14]
There are several arithmetic, string, array, and bitwise operators.

Expressions in SQL Statements [page 34]
An expression is a statement that can be evaluated to return values.

Search Conditions [page 58]
A search condition is the criteria specified for a WHERE clause, a HAVING clause, a CHECK clause, an ON phrase in a join, or an IF expression. A search condition is also called a predicate.

Special Values [page 87]
Special values can be used in expressions, and as column defaults when creating tables.

%TYPE and %ROWTYPE Attributes [page 115]
In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

SQL Variables [page 123]
The supported variables can be grouped by scope: connection, database, and global.

Comments [page 127]
Comments are used to attach explanatory text to SQL statements or statement blocks. The database server does not execute comments.

Named Parameters [page 128]
Functions and procedures that are referenced from the CALL statement, the EXECUTE statement (Transact-SQL), the FROM clause of a DML statement, and the TRIGGER EVENT statement support positional parameters and named parameters. Named parameters support specifying any subset of the available parameters in any order.

1.1.1 Keywords

Each SQL statement contains one or more keywords. SQL is case insensitive to keywords, but throughout the documentation, keywords are indicated in uppercase.

For example, in the following statement, SELECT and FROM are keywords:

```
SELECT *
   FROM Employees;
```

The following statements are equivalent to the one above:

```
Select *
   From Employees;
select * from Employees;
sELECT * FRoM Employees;
```

Some keywords cannot be used as identifiers without surrounding them in double quotes, square brackets, or back quotes (`...`). These are called reserved words. Other keywords, such as DBA, do not require quotation marks, and are not reserved words.

In this section:

Reserved Words [page 6]
Some keywords in SQL are considered reserved words.

1.1.1.1 Reserved Words

Some keywords in SQL are considered reserved words.

Reserved words are words that must be treated specially when used in SQL syntax. Many of the keywords that appear in SQL statements are reserved words (for example, the word select. To use a reserved word in a SQL statement as an identifier, enclose it in double quotes, square brackets or back quotes .For example, you use the following syntax to retrieve the contents of a table named SELECT.

```
SELECT *
   FROM "SELECT"
```

To obtain the list of reserved words, use the sa_reserved_words system procedure. For example:

```
SELECT * FROM sa_reserved_words() ORDER BY reserved_word;
```

Other notes regarding reserved words:

- SQL keywords are not case sensitive and the following words may appear in uppercase, lowercase, or any combination of the two. All strings that differ only in capitalization from one of the following words are reserved words.
- You can turn off keyword restrictions using the non_keywords option.
- The reserved_keywords option turns on individual keywords that are disabled by default.
- If you are using Embedded SQL, you can use the sql_needs_quotes database library function to determine whether a string requires quotation marks. A string requires quotes if it is a reserved word or if it contains a character not ordinarily allowed in an identifier.

Related Information

sql_needs_quotes Function
non_keywords Option
reserved_keywords Option
sa_reserved_words System Procedure [page 1687]

1.1.2 Identifiers

Identifiers are the names of objects in the database, such as user IDs, tables, and columns.

Identifiers have a maximum length of 128 bytes and are composed from alphabetic characters and digits, as well as the underscore character (_), at sign (@), number sign (#), and dollar sign ($). Leading digits are allowed but the identifier must be quoted. Other special characters are allowed but the identifier must be quoted. The database collation sequence dictates which characters are considered alphabetic or digit characters.

The following characters are not permitted in identifiers:

- Double quotes

- Control characters (characters with an ordinal value of less than 32, or the character value 127)
- Backslashes
- Square brackets
- Back quotes

> **i** Note
>
> If you are reloading a database that is of an earlier version than 16.0, then remove any square brackets or back quotes in identifiers; otherwise, the reload fails.

The following characters are not permitted in identifiers used as user or role names:

- Leading or trailing whitespace
- Leading single quote
- Semicolons

If the quoted_identifier database option is set to Off, then double quotes delimit SQL strings and cannot be used to delimit identifiers. However, you can use square brackets or back quotes to delimit identifiers, regardless of the setting of quoted_identifier. The default setting for the quoted_identifier option is Off for Open Client and jConnect connections; otherwise, the default is On.

An indirect identifier can also be substituted for an identifier in a statement. Indirect identifiers allow you to specify the name of a variable that stores an identifier, instead of specifying the identifier directly. However, indirect identifiers are only supported for a selection of objects and there are restrictions on how they can be used.

Quoting Identifiers

If any of the following conditions are true, then always enclose an identifier in double quotes, square brackets, or back quotes (`...`):

- The identifier contains leading, trailing, or embedded spaces.
- The first character of the identifier is not an alphabetic character, the underscore character (_), at sign (@), number sign (#), or dollar sign ($). For example, the first character is a digit.
- The identifier contains characters other than the alphabetic characters, digits, underscore character (_), at sign (@), number sign (#), and dollar sign ($).
- The identifier is a reserved word.

For compatibility with other database management systems, it is recommended that you avoid the use of special characters in identifier names, including but not limited to any of the following:

- Leading or trailing whitespace
- Leading single quote
- Semicolons

Standards

ANSI/ISO SQL Standard

The ability to create identifiers of up to 128 characters is optional ANSI/ISO SQL Language Feature F391.

Example

The following strings are all valid identifiers:

- Surname
- "Client Name"
- `Client Name`
- [Surname]
- SomeBigName
- `[@myVar]`

In this section:

Related Information

Reserved Words [page 6]
quoted_identifier Option
Indirect Identifiers [page 8]
TABLE REF Data Type [page 188]

1.1.2.1 Indirect Identifiers

Use indirect identifiers when the name of an object must be determined at statement run time, or to avoid exposing the names of underlying objects in a statement.

Syntax

Specify indirect identifiers in statements by enclosing them in square brackets and back quotes (for example, `` `[@myVariable]` ``), where @myVariable is the name of an existing variable that stores the name of the actual object you are operating on.

Remarks

When an identifier, A, in a statement specifies a variable that contains another identifier, B, identifier A is called an **indirect identifier**. If you use EXECUTE IMMEDIATE to dynamically construct statements inside procedures (specifically, if you are substituting identifiers in your DML EXECUTE IMMEDIATE statements), then consider using indirect identifiers instead. Indirect identifiers are a safer practice than using EXECUTE IMMEDIATE in your application logic.

Building indirect identifiers into your application logic improves the dynamic capability of your product. For example, suppose your application periodically creates a table using the table creation time stamp information as part of the identifier for the table (for example, CurrentOrders023003032015, where 023003032015 is the time stamp when the table was created). Now suppose that your application has a procedure that needs to query this dynamically named table. You could declare a variable called @currentOrders to store the table name, and then update the variable each time the table is created. Then, you could modify your procedure to include an indirect identifier for the table (`` `[@currentOrders]` ``). When the procedure is called, the indirect identifier is replaced with the value of the variable and the procedure runs as though the actual table name was specified.

Indirect identifiers are supported in SELECT statements, procedure and function calls, and DML statements as a substitute for an explicit identifier for the following objects:

- tables
- columns
- mutexes
- semaphores
- user IDs when specified as object owners (for example `owner.object-name`)

Indirect identifiers are also supported in statements that change the status of mutexes and semaphores (for example, WAITFOR SEMAPHORE statement).

Before a statement is executed, an indirect identifier is replaced by the value stored in the variable being referenced, and privilege checking is performed.

Indirect identifier values have a maximum length of 128 bytes and can be of type CHAR, VARCHAR, or LONG VARCHAR.

For statements where an identifier is required, if the indirect identifier used to specify the name of a column or table is NULL, an empty string, or another invalid name, then the result is in an error. However, an indirect identifier may be NULL if it is used as an optional part of a qualified name, such as the owner of a table. A NULL for an optional part of the identifier is treated as though it is absent.

An indirect identifier replaces one portion of an identifier; it cannot replace the full identifier specification. For example, if you have a variable @var set to `'GROUPO.Employees'`, then an error is returned if you attempt to perform a SELECT operation on the GROUPO.Employees table by using an indirect identifier (for example, `SELECT * FROM `[@var]``). Instead, you must create a variable to store the user portion of the name, and then reference both objects using indirect references (for example, `` `[@owner]`.`[@var]` ``)

Use of indirect identifiers overlaps with the use of table reference variables; both are ways of indirectly referring to a table. However, a table reference variable can provide access to a table that is not accessible in the current context, whereas an indirect identifier cannot. Additionally, table reference values are resolved at creation time, while indirect identifiers are resolved at run time.

Privileges and Permissions

Privileges on the objects being indirectly referenced in a statement are checked at the time that the indirect identifiers are evaluated and are enforced prior to the statement execution.

Examples

Many of the examples below show the variables being created using the CREATE VARIABLE statement; this was done to make the examples easy to try in Interactive SQL. However, a more likely scenario is that you would declare a variable (DECLARE statement) within the scope of a procedure or function, and then reference the variable as part of an indirect identifier in a subsequent statement within that procedure, or pass variables in as parameters.

The following example creates a variable called @col to hold the name of a column (Surname) in the GROUPO.Employees table. The SELECT statement queries the contents of the Employees.Surname column by specifying an indirect identifier (`` `[@col]` ``):

```
CREATE OR REPLACE VARIABLE @col LONG VARCHAR = 'Surname';
SELECT E.`[@col]` FROM GROUPO.Employees E;
```

The following example shows how to use indirect identifiers to query a table:

```
CREATE OR REPLACE VARIABLE t_owner LONG VARCHAR = 'GROUPO';
CREATE OR REPLACE VARIABLE t_name LONG VARCHAR = 'Employees';
SELECT * FROM `[t_owner]`.`[t_name]`;
```

The following example creates a procedure with an IN parameter (@tableref) that takes a table reference, an IN parameter (@columnname) that takes the name of a column, and an IN parameter (@value) that takes the integer column value of the user to delete. The body of the procedure defines how the parameters will be used to delete the required record from the table.

```
CREATE PROCEDURE mydelete( IN @tableref TABLE REF,
                           IN @columnname LONG VARCHAR,
                           IN @value INT )
NO RESULT SET
SQL SECURITY INVOKER
BEGIN
    DELETE FROM TABLE REF(@tableref) AS T WHERE T.`[@columnname]` = @value;
END;
CALL mydelete( TABLE REF(Employees), 'EmployeeID', 102);
CALL mydelete( TABLE REF(Contacts), 'ID',  11);
```

In the first CALL statement, the database server searches the Employee.EmployeeID column for a row that matches the value stored in the @value parameter (102), and then deletes the row. In the second CALL statement, the database server searches the Contacts.ID column for a row that matches the value stored in the @value parameter (11), and then deletes the row.

Related Information

SQL Variables [page 123]

TABLE REF Data Type [page 188]
Identifiers [page 6]

1.1.3 Strings

A string is a sequence of characters up to 2 GB in size.

A string can occur in SQL:

- as a **string literal**. A string literal is a sequence of characters enclosed in single quotes (apostrophes). A string literal represents a particular, constant value, and it may contain escape sequences for special characters that cannot be easily typed as characters.
- as the value of a column or variable with a CHAR or NCHAR data type.
- as the result of evaluating an expression.

The length of a string can be measured in two ways:

Byte length

The byte length is the number of bytes in the string.

Character length

The character length is the number of characters in the string, and is based on the character set being used.

For single-byte character sets, such as cp1252, the byte-length and character-length are the same. For multibyte character sets, a string's byte-length is greater than or equal to its character-length.

Related Information

String Literals [page 13]

1.1.4 Constants

Binary literals and string literals can be used as constants.

In this section:

Binary Literals [page 12]
A binary literal is a sequence of hexadecimal characters consisting of digits 0-9 and uppercase and lowercase letters A-F.

String Literals [page 13]
A string literal is a sequence of characters enclosed in single quotes.

1.1.4.1 Binary Literals

A binary literal is a sequence of hexadecimal characters consisting of digits 0-9 and uppercase and lowercase letters A-F.

Hexadecimal constants in the form of 0x12345678 are treated as binary strings. An unlimited number of digits can be added after the 0x.

A binary literal is sometimes referred to as a binary constant. The preferred term is binary literal.

In this section:

1.1.4.1.1 Converting to and from Hexadecimal Values

You can use the CAST, CONVERT, HEXTOINT, and INTTOHEX functions to convert a binary string to an integer.

The CAST and CONVERT functions convert hexadecimal constants to TINYINT, signed and unsigned 32-bit integer, signed and unsigned 64-bit integer, NUMERIC, and so on. The HEXTOINT function only converts a hexadecimal constant to a signed 32-bit-integer.

The value returned by the CAST function cannot exceed 8 digits. Values exceeding 8 digits return an error. Zeros are added to the left of values less than 8 digits. For example, the following argument returns the value -2,147,483,647:

```
SELECT CAST ( 0x0080000001 AS INT );
```

The following argument returns an error because the 10-digit value cannot be represented as a signed 32-bit integer:

```
SELECT CAST ( 0xff80000001 AS INT );
```

The value returned by the HEXTOINT function can exceed 8 digits if the value can be represented as a signed 32-bit integer. The HEXTOINT function accepts string literals or variables consisting only of digits and the uppercase or lowercase letters A-F, with or without a 0x prefix. The hexadecimal value represents a negative integer when the 8th digit from the right is one of the digits 8-9, the uppercase or lowercase letters A-F, or the previous leading digits are all uppercase or lowercase letter F.

The following arguments return the value -2,147,483,647:

```
SELECT HEXTOINT( '0xFF80000001' );
```

```
SELECT HEXTOINT( '0x80000001' );
```

```
SELECT HEXTOINT ( '0xFFFFFFFFFFFFFFFF80000001' );
```

The following argument returns an error because the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

```
SELECT HEXTOINT( '0x0080000001' );
```

Related Information

CAST Function [Data Type Conversion] [page 275]
CONVERT Function [Data Type Conversion] [page 294]
HEXTOINT Function [Data Type Conversion] [page 400]
INTTOHEX Function [Data Type Conversion] [page 424]

1.1.4.2 String Literals

A string literal is a sequence of characters enclosed in single quotes.

For example, `'Hello world'` is a string literal of type CHAR. Its byte length is 11, and its character length is also 11.

A string literal is sometimes referred to as a string constant, literal string, or just as a string. The preferred term is string literal.

You can specify an NCHAR string literal by prefixing the quoted value with N. For example, `N'Hello world'` is a string literal of type NCHAR. Its byte length is 11, and its character length is 11. The bytes within an NCHAR string literal are interpreted using the database's CHAR character set, and then converted to NCHAR. The syntax `N'string'` is a shortened form for `CAST( 'string' AS NCHAR )`.

In this section:

Escape Sequences [page 13]
Sometimes you must put characters into string literals that cannot be typed or entered normally. Examples include control characters (such as a new line character), single quotes (which would otherwise mark the end of the string literal), and hexadecimal byte values. For this purpose, you use an escape sequence.

1.1.4.2.1 Escape Sequences

Sometimes you must put characters into string literals that cannot be typed or entered normally. Examples include control characters (such as a new line character), single quotes (which would otherwise mark the end of the string literal), and hexadecimal byte values. For this purpose, you use an escape sequence.

The following examples show how to use escape sequences in string literals.

- A single quote is used to mark the beginning and end of a string literal, so a single quote in a string must be escaped using an additional single quote, as follows: `'John''s database'`

- A backslash followed by any character other than n, x, X, or \ is interpreted as two separate characters. For example, \q inserts a backslash and the letter q.
  Hexadecimal escape sequences can be used for any character or binary value. A hexadecimal escape sequence is a backslash followed by an x followed by two hexadecimal digits. The hexadecimal value is interpreted as a character in the CHAR character set for both CHAR and NCHAR string literals. The value \x09 must be coded as \\x09 if you don't want the value stored as a single tab character, but \xyy would be stored as \xyy. The following example, in code page 1252, represents the digits 1, 2, and 3, followed by the euro currency symbol: `'123\x80'`.
- Escape a backslash character by using an additional backslash, as follows: `'c:\\november'`. For paths, you can also use the forward slash (/) instead of a backslash: `'c:/november'`.
- Represent a new line character by using a backslash followed by n (\n), specify: `'First line:\nSecond line:'`

You can use the same characters and escape sequences with NCHAR string literals as with CHAR string literals.

To use Unicode characters that cannot be typed directly into the string literal, use the UNISTR function.

Related Information

UNISTR Function [String] [page 609]

1.1.5 Operators

There are several arithmetic, string, array, and bitwise operators.

The normal precedence of operations applies. Expressions in parentheses are evaluated first, then multiplication and division before addition and subtraction. String concatenation happens after addition and subtraction.

In this section:

Join Operators [page 33]
SQL Anywhere supports two additional comparison operators, *= and =*, which are the Transact-SQL outer join operators.

Operator Precedence [page 34]
The precedence of operators in expressions is significant.

Related Information

Search Conditions [page 58]

1.1.5.1 Comparison Operators

You can use comparison operators to compare values.

The syntax for comparison is as follows:

```
expression comparison-operator expression
```

where `comparison-operator` is one of the following:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
!>	Not greater than
!<	Not less than

Case Sensitivity

When you create a database, you indicate whether string comparisons are case sensitive or not.

By default, databases are created case insensitive. For example, 'Dirk' is the same as 'DIRK'.

You can find out the database case sensitivity using the Information utility (dbinfo):

```
dbinfo -c "uid=DBA;pwd=sql"
```

Look for the collation *CaseSensitivity* information.

You can also ascertain the database case sensitivity from SQL Central using the *Settings* tab of the *Database Properties* window.

Comparing dates

When comparing dates, < means earlier and > means later.

Sort order

When you create a database, you chose the database collations for CHAR and NCHAR data types.

When comparing character data, < means earlier in the sort order and > means later in the sort order. The sort order is determined by the database collation.

You can find out the database collation using the Information utility (dbinfo):

```
dbinfo -c "uid=DBA;pwd=sql"
```

You can also ascertain the collation from SQL Central using the *Settings* tab of the *Database Properties* window.

Trailing blanks

When you create a database, you indicate whether trailing blanks are ignored for comparison purposes.

By default, databases are created with trailing blanks not ignored. For example, 'Dirk' is not the same as 'Dirk '. You can create databases with blank padding, so that trailing blanks are ignored.

You can find out the database blank padding property using the Information utility (dbinfo):

```
dbinfo -c "uid=DBA;pwd=sql"
```

You can also ascertain the database blank padding property from SQL Central by inspecting the *Ignore trailing blanks* property in the *Settings* tab of the *Database Properties* window.

Related Information

Transact-SQL-compatible Databases
Troubleshooting: Case Sensitivity and Remote Data Access

1.1.5.2 Logical Operators

Expressions can be combined, negated, or tested using logical operators.

For example, search conditions can be combined using the AND or OR operators. You can also negate them using the NOT operator, or test whether an expression would evaluate to true, false, or unknown, using the IS operator.

AND operator

The AND operator is placed between search conditions as follows:

```
...WHERE condition1 AND condition2
```

When using AND, the combined condition is TRUE if both conditions are TRUE, FALSE if either condition is FALSE, and UNKNOWN otherwise.

OR operator

The OR operator is placed between search conditions as follows:

```
...WHERE condition1 OR condition2
```

When using OR, the combined condition is TRUE if either condition is TRUE, FALSE if both conditions are FALSE, and UNKNOWN otherwise.

NOT operator

The NOT operator is placed before a condition to negate the condition, as follows:

```
...WHERE NOT condition
```

The NOT condition is TRUE if `condition` is FALSE, FALSE if `condition` is TRUE, and UNKNOWN if `condition` is UNKNOWN.

IS operator

The IS operator is placed between an expression and the truth value you are testing for. The syntax for the IS operator is as follows:

```
expression IS [ NOT ] truth-value
```

The IS condition is TRUE if the `expression` evaluates to the supplied `truth-value`, which must be one of TRUE, FALSE, UNKNOWN, or NULL. Otherwise, the value is FALSE.

For example, `5*3=15 IS TRUE` tests whether the expression `5*3=15` evaluates to TRUE.

Related Information

Three-valued Logic [page 85]

1.1.5.3 Arithmetic Operators

You can use arithmetic operators to perform arithmetic operations on expressions.

The software supports the following arithmetic operators:

expression + expression

Addition. If either expression is the NULL value, the result is NULL.

expression - expression

Subtraction. If either expression is the NULL value, the result is NULL.

-expression

Negation. If the expression is the NULL value, the result is NULL.

expression * expression

Multiplication. If either expression is NULL, the result is NULL.

expression / expression

Division. If either expression is NULL or if the second expression is 0, the result is NULL.

expression % expression

Modulo finds the integer remainder after a division involving two whole numbers. For example, 21 % 11 = 10 because 21 divided by 11 equals 1 with a remainder of 10.

Support for Arithmetic Operators on Date and Time Data Types

The support of arithmetic operators on date and time data types is limited to + and -. There are some constraints on how these operators can be used.

For an operation, `expression1 + expression2`, one of the expressions must be a date, and the other expression must be a time. The result is a TIMESTAMP formed by combining the date and time expressions.

For an operation, `expression1 - expression2`, the restrictions and behavior are as follows:

expression1 is a subtype of TIMESTAMP, and expression2 is a subtype of SIGNED LONG

The result is a TIMESTAMP with `expression2` days subtracted.

expression1 is a subtype of TIMESTAMP, and so is expression2

The result is the number of days between `expression1` and `expression2`, which could also be expressed as `( DATEDIFF ( day, expression1,expression2 ) )`.

expression1 is a subtype of TIMESTAMP, and expression2 is a subtype of NUMERIC or VARCHAR

`expression2` is first converted to a NUMERIC and is then interpreted as a number of days. The result is the number of days between `expression1` and `expression2`, which could also be expressed as `( DATEDIFF ( day, expression1 , expression2 ) )`.

Standards

ANSI/ISO SQL Standard

The use of % as a modulo operator is not in the standard.

1.1.5.4 String Operators

You can use string operators to concatenate strings.

The software supports the following string operators:

expression || expression

String concatenation (two vertical bars). If either string is NULL, it is treated as the empty string for concatenation.

expression + expression

Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

For example, the following query returns the integer value 579:

```
SELECT 123 + 456;
```

whereas the following query returns the character string 123456:

```
SELECT '123' + '456';
```

Standards

ANSI/ISO SQL Standard

The || operator is the ANSI/ISO SQL Standard string concatenation operator. However, in the SQL standard, if either operand of || is the NULL value, then the result of the concatenation is also NULL. In the software, the || operator treats NULL as an empty string.

In this section:

OPENXML Operator [page 20]
Generates a result set from an XML document.

1.1.5.4.1 OPENXML Operator

Generates a result set from an XML document.

Syntax - Specify the XML

```
OPENXML(
xml-data
, xpath
[, flags
[, namespaces ] ]
)
WITH ( column-name column-type
[ xpath ] [ , ... ]
)
```

Syntax - Specify a File Containing the XML

```
OPENXML( { USING FILE | USING VALUE }
xml-data
, xpath
[, flags
[, namespaces ] ]
)
WITH ( column-name column-type
[ xpath ] [ , ... ]
)
[ OPTION ( scan-option ) ]
[ AS ] correlation-name
```

```
scan-option :
ENCODING encoding
| BYTE ORDER MARK { ON | OFF }
```

Parameters

WITH clause

Specifies the schema of the result set and how the value is found for each column in the result set. WITH clause `xpath` arguments are matched relative to the matches for the `xpath` in the second argument. If a WITH clause expression matches more than one node, then only the first node in the document order is used. If the node is not a text node, then the result is found by appending all the text node descendants. If a WITH clause expression does not match any nodes, then the column for that row is NULL.

The `xpath` arguments in the WITH clause can be literal strings or variables. See http://www.w3.org/TR/xpath .

The OPENXML WITH clause syntax is similar to the syntax for selecting from a stored procedure.

USING FILE | USING VALUE

Use the USING FILE clause to load data from a file.

Use the USING VALUE clause to load data from any expression of CHAR, NCHAR, BINARY, or LONG BINARY type, or BLOB string.

xml-data

The XML on which the result set is based. This can be any string expression, such as a constant, variable, or column.

The `xml-data` is parsed directly in the NCHAR encoding if there are any NCHAR columns in the output. The xpath and namespaces arguments are also converted and parsed in the NCHAR encoding.

xpath

A string containing an XPath query. XPath allows you to specify patterns that describe the structure of the XML document you are querying. The XPath pattern included in this argument selects the nodes from the XML document. Each node that matches the XPath query in the second `xpath` argument generates one row in the table.

Metaproperties can only be specified in WITH clause `xpath` arguments. A metaproperty is accessed within an XPath query as if it was an attribute. If a `namespaces` is not specified, then by default the prefix mp is bound to the Uniform Resource Identifier (URI) urn:sap-com:sa-xpath-metaprop. If a `namespaces` is specified, this URI must be bound to mp or some other prefix to access metaproperties in the query. Metaproperty names are case sensitive. The OPENXML statement supports the following metaproperties:

@mp:id

returns an ID for a node that is unique within the XML document. The ID for a given node in a given document may change if the database server is restarted. The value of this metaproperty increases with document order.

@mp:localname

returns the local part of the node name, or NULL if the node does not have a name.

@mp:prefix

returns the prefix part of the node name, or NULL if the node does not have a name or if the name is not prefixed.

@mp:namespaceuri

returns the URI of the namespace that the node belongs to, or NULL if the node is not in a namespace.

@mp:xmltext

returns a subtree of the XML document in XML form. For example, when you match an internal node, you can use this metaproperty to return an XML string, rather than the concatenated values of the descendant text nodes.

flags

Indicates the mapping that should be used between the XML data and the result set when an XPath query is not specified in the WITH clause. If the `flags` parameter is not specified, the default behavior

is to map attributes to columns in the result set. The `flags` parameter can have one of the following values:

Value	Description
1	XML attributes are mapped to columns in the result set (the default).
2	XML elements are mapped to columns in the result set.

namespace-declaration

An XML document. The in-scope namespaces for the query are taken from the root element of the document. If namespaces are specified, then you must include a `flags` argument, even if all the `xpath` arguments are specified.

column-name

The name of the column in the result set.

column-type

The data type of the column in the result set. The data type must be compatible with the values selected from the XML document.

OPTION clause

Use the OPTION clause to specify parsing options to use for the input file, such as escape characters, delimiters, encoding, and so on.

ENCODING clause

The ENCODING clause allows you to specify the encoding that is used to read the file.

If the ENCODING clause is not specified, then encoding for values is assumed to be in the database character set (db_charset) if the values are of type CHAR or BINARY, and NCHAR database character set (nchar_charset) if the values are of type NCHAR.

BYTE ORDER MARK clause

Use the BYTE ORDER MARK clause to specify whether a byte order mark (BOM) is present in the encoding. By default, this option is ON, which enables the server to search for and interpret a byte order mark (BOM) at the beginning of the data. If BYTE ORDER MARK is OFF, the server does not search for a BOM.

You must specify the BYTE ORDER MARK clause if the input data is encoded.

If the ENCODING clause is specified:

- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to verify the endianness of the data. If you specify the wrong endian, an error is returned.
- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.
- If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, the database server searches for a BOM at the beginning of the data. If a BOM is present it is ignored.

If the ENCODING clause is not specified:

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, the server looks for a BOM at the beginning of the input data. If a BOM is located, the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or UTF-8) and the BOM is not considered to be part of the data to be loaded.
- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, or a BOM is not found at the beginning of the input data, the database CHAR encoding is used.

Remarks

The OPENXML operator parses the `xml-data` and models the result as a tree. The tree contains a separate node for each element, attribute, and text node, or other XML construct. The XPath queries supplied to the OPENXML operator are used to select nodes from the tree, and the selected nodes are then mapped to the result set.

The XML parser used by the OPENXML operator is non-validating, and does not read the external DTD subset or external parameter entities.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

When there are multiple matches for a column expression, the first match in the document order (the order of the original XML document before it was parsed) is used. NULL is returned if there are no matching nodes. When an internal node is selected, the result is all the descendant text nodes of the internal node concatenated together.

Columns of type BINARY, LONG BINARY, IMAGE, and VARBINARY are assumed to be in base64-encoded format and are decoded automatically. If you generate XML using the FOR XML clause, these types are base64-encoded, and can be decoded using the OPENXML operator.

The OPENXML operator supports a subset of the XPath syntax, as follows:

- The child, self, attribute, descendant, descendant-or-self, and parent axes are fully supported.
- Both abbreviated and unabbreviated syntax can be used for all supported features. For example, `'a'` is equivalent to `'child::a'` and `'..'` is equivalent to `'parent::node()'`.
- Name tests can use wildcards. For example, `'a/*/b'`.
- The following kind tests are supported: node(), text(), processing-instruction(), and comment().
- Qualifiers of the form `expr1`[`expr2`] and `expr1`[`expr2`="`string`"] can be used, where `expr2` is any supported XPath expression. A qualifier evaluates TRUE if `expr2` matches one or more nodes. For example, `'a[b]'` finds `a` nodes that have at least one `b` child, and `a[b="I"]` finds `a` nodes that have at least one `b` child with a text value of `I`.

Privileges

If the USING FILE clause is specified, you must have the READ FILE system privilege. Otherwise, no privileges are required.

Example

The following query generates a result set from the XML document supplied as the first argument to the OPENXML operator:

```
SELECT * FROM OPENXML( '<products>
                <ProductType ID="301">Tee Shirt</ProductType>
                <ProductType ID="401">Baseball Cap</ProductType>
                </products>',
                '/products/ProductType' )
WITH ( ProductName LONG VARCHAR 'text()', ProductID CHAR(3) '@ID');
```

This query generates the following result:

ProductName	ProductID
Tee Shirt	301
Baseball Cap	401

In the following example, the first <ProductType> element contains an entity. When you execute the query, this node is parsed as an element with four children: Tee, &, Sweater, and Set. You can use a period (.) to concatenate the children together in the result set.

```
SELECT * FROM OPENXML( '<products>
                <ProductType ID="301">Tee &amp; Sweater Set</ProductType>
                <ProductType ID="401">Baseball Cap</ProductType>
                </products>',
                '/products/ProductType' )
WITH ( ProductName LONG VARCHAR '.', ProductID CHAR(3) '@ID');
```

This query generates the following result:

ProductName	ProductID
Tee & Sweater Set	301
Baseball Cap	401

The following query uses an equality predicate to generate a result set from the supplied XML document.

```
SELECT * FROM OPENXML('<EmployeeDirectory>
   <Employee>
      <column name="EmployeeID">105</column>
      <column name="GivenName">Matthew</column>
      <column name="Surname">Cobb</column>
      <column name="Street">7 Pleasant Street</column>
      <column name="City">Grimsby</column>
      <column name="State">UT</column>
      <column name="PostalCode">02154</column>
      <column name="Phone">6175553840</column>
   </Employee>
  <Employee>
      <column name="EmployeeID">148</column>
      <column name="GivenName">Julie</column>
      <column name="Surname">Jordan</column>
      <column name="Street">1244 Great Plain Avenue</column>
      <column name="City">Woodbridge</column>
      <column name="State">AZ</column>
      <column name="PostalCode">01890</column>
      <column name="Phone">6175557835</column>
   </Employee>
```

```
  <Employee>
      <column name="EmployeeID">160</column>
      <column name="GivenName">Robert</column>
      <column name="Surname">Breault</column>
      <column name="Street">358 Cherry Street</column>
      <column name="City">Milton</column>
      <column name="State">PA</column>
      <column name="PostalCode">02186</column>
      <column name="Phone">6175553099</column>
   </Employee>
  <Employee>
      <column name="EmployeeID">243</column>
      <column name="GivenName">Natasha</column>
      <column name="Surname">Shishov</column>
      <column name="Street">151 Milk Street</column>
      <column name="City">Grimsby</column>
      <column name="State">UT</column>
      <column name="PostalCode">02154</column>
      <column name="Phone">6175552755</column>
   </Employee>
</EmployeeDirectory>', '/EmployeeDirectory/Employee')
WITH ( EmployeeID INT 'column[@name="EmployeeID"]',
       GivenName    CHAR(20) 'column[@name="GivenName"]',
       Surname      CHAR(20) 'column[@name="Surname"]',
       PhoneNumber  CHAR(10) 'column[@name="Phone"]');
```

This query generates the following result set:

EmployeeID	GivenName	Surname	PhoneNumber
105	Matthew	Cobb	6175553840
148	Julie	Jordan	6175557835
160	Robert	Breault	6175553099
243	Natasha	Shishov	6175552755

The following query uses the XPath @attribute expression to generate a result set:

```
SELECT * FROM OPENXML( '<Employee
      EmployeeID="105"
      GivenName="Matthew"
      Surname="Cobb"
      Street="7 Pleasant Street"
      City="Grimsby"
      State="UT"
      PostalCode="02154"
      Phone="6175553840"
      />', '/Employee' )
WITH ( EmployeeID   INT      '@EmployeeID',
       GivenName    CHAR(20) '@GivenName',
       Surname      CHAR(20) '@Surname',
       PhoneNumber  CHAR(10) '@Phone');
```

The following query operates on an XML document like the one used in the above query, except that an XML namespace has been introduced. It demonstrates the use of wildcards in the name test for the XPath query, and generates the same result set as the above query.

```
SELECT * FROM OPENXML( '<Employee  xmlns="http://www.sap.com/EmployeeDemo"
      EmployeeID="105"
      GivenName="Matthew"
      Surname="Cobb"
      Street="7 Pleasant Street"
      City="Grimsby"
```

```
      State="UT"
      PostalCode="02154"
      Phone="6175553840"
/>', '/*:Employee' )
WITH ( EmployeeID   INT      '@EmployeeID',
       GivenName    CHAR(20) '@GivenName',
       Surname      CHAR(20) '@Surname',
       PhoneNumber  CHAR(10) '@Phone');
```

Alternatively, you could specify a namespace declaration:

```
SELECT * FROM OPENXML( '<Employee xmlns="http://www.sap.com/EmployeeDemo"
      EmployeeID="105"
      GivenName="Matthew"
      Surname="Cobb"
      Street="7 Pleasant Street"
      City="Grimsby"
      State="UT"
      PostalCode="02154"
      Phone="6175553840"
/>', '/prefix:Employee', 1, '<r xmlns:prefix="http://www.sap.com/EmployeeDemo"/
>' )
WITH ( EmployeeID   INT      '@EmployeeID',
       GivenName    CHAR(20) '@GivenName',
       Surname      CHAR(20) '@Surname',
       PhoneNumber  CHAR(10) '@Phone');
```

The following example illustrates the USING FILE syntax.

```
SELECT Products.* FROM OPENXML( USING FILE 'products.xml', '/products/
ProductType' )
WITH ( ProductName LONG VARCHAR 'text()', ProductID CHAR(3) '@ID' ) AS Products;
```

The products.xml file contains the following XML text.

```
<products>
<ProductType ID="301">Tee Shirt</ProductType>
<ProductType ID="401">Baseball Cap</ProductType>
</products>
```

This query generates the following result:

ProductName	ProductID
Tee Shirt	301
Baseball Cap	401

The following example illustrates the USING VALUE syntax.

```
CREATE OR REPLACE VARIABLE xmltext LONG VARCHAR =
'<feed>
  <title>Example Feed</title>
  <link href="http://example.org/"/>
  <updated>2017-04-27T14:30:04Z</updated>
  <author>
    <name>John Doe</name>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
  <entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2016/12/13/atom03"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2016-12-13T18:30:02Z</updated>
```

```
    <summary>Some text.</summary>
  </entry>
  <entry>
    <title>More Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2017/04/27/atom04"/>
    <id>urn:uuid:1325c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2017-04-27T13:29:12Z</updated>
    <summary>Some more text.</summary>
  </entry>
</feed>';
SELECT * FROM OPENXML( USING VALUE xmltext, '/feed/entry' )
WITH ( Title        LONG VARCHAR '../title',
       Link         LONG VARCHAR '../link/@href',
       Updated      LONG VARCHAR '../updated',
       Author       LONG VARCHAR '../author/name',
       Ident        LONG VARCHAR '../id',
       EntryTitle   LONG VARCHAR 'title',
       EntryLink    LONG VARCHAR 'link/@href',
       EntryId      LONG VARCHAR 'id',
       EntryUpdated LONG VARCHAR 'updated',
       EntrySummary LONG VARCHAR 'summary') AS F;
```

This query generates a result with 2 rows.

Related Information

FOR XML and Binary Data
XML Import Using the OPENXML Operator
Supported Character Sets
SQL Data Types [page 129]
FROM Clause [page 1173]

1.1.5.5 Array Operators

You can use array operators to concatenate arrays.

The software supports the following string operator:

expression || expression

Array concatenation (two vertical bars). If either array is NULL, it is treated as a zero-length array for concatenation.

Standards

ANSI/ISO SQL Standard

The || operator is the ANSI/ISO SQL Standard concatenation operator. However, in the SQL standard, if either operand of || is the NULL value, then the result of the concatenation is also NULL. In the software, the || operator treats NULL as a zero-length array.

In this section:

1.1.5.5.1 UNNEST Array Operator

Creates a derived table from the given array expressions that results in one row per array element.

Syntax

```
UNNEST ( array-expression [, ...] )
[ WITH ORDINALITY ]
```

Parameters

array-expression

An array to derive a table column from.

WITH ORDINALITY

The WITH ORDINALITY clause permits the application to recall the original array element from which each value was obtained. Valid UNNEST derived tables must have names specified (by using the AS clause) for each of the resulting expressions. The order of the resulting rows from unnest is not guaranteed. Users can achieve a desired ordering with an ORDER BY clause.

Remarks

If the array expressions have different cardinalities, the missing output expressions from the shorter array(s) are set to NULL. If the WITH ORDINALITY clause is specified, the result set contains an integer column that identifies the array element's cardinal number that the row represents. The new column is appended to the unnest derived table as its last column.

Privileges

None

Example

The following example illustrates how to use of the unnest operator with two arrays that have different cardinalities:

```
SELECT * FROM UNNEST( ARRAY(2,3,4), ARRAY(4,5,6) ) WITH ORDINALITY AS DT(X,Y,Z);
```

The SQL statement returns the following result:

X	Y	Z
2	4	1
3	5	2
4	6	3

Examples

The following statements create a simple array and populate it with data:

```
CREATE OR REPLACE VARIABLE x1 ARRAY (10) OF INT;
SELECT ARRAY_AGG(id) INTO x1 FROM GROUPO.Products;
```

The following statement returns the data in the array, unnested:

```
SELECT * FROM UNNEST(x1) AS DT(X);
```

X
300
301
302
400
401
500
501
600
601
700

The following statement returns the same data, but adds ordinality (Y column):

```
SELECT * FROM UNNEST(x1) WITH ORDINALITY AS DT(X,Y);
```

X	Y
300	1
301	2
302	3
400	4
401	5
500	6
501	7
600	8
601	9
700	10

The following statement returns the data from cell 1 of the array:

```
SELECT x1[[1]];
```

x1[[1]]
300

The following statements create a two dimensional array and populate it with data:

```
CREATE OR REPLACE VARIABLE x1 ARRAY(2) OF ARRAY (10) OF INT;
SELECT ARRAY_AGG( "id" ) INTO x1[[1]] FROM GROUPO.Products;
SELECT ARRAY_AGG( GROUPO.Products.Quantity ) INTO x1[[2]] FROM GROUPO.Products;
```

The following statement returns the data in the array, unnested:

```
SELECT * FROM UNNEST( x1[[1]], x1[[2]] ) WITH ORDINALITY AS DT( X, Y, Z);
```

X	Y	Z
300	28	1
301	54	2
302	75	3
400	112	4
401	12	5

X	Y	Z
500	36	6
501	28	7
600	39	8
601	32	9
700	80	10

The following statement returns the data found in the second column first row of the array::

```
SELECT (x1[[2]])[[1]];
```

(x1[[2]])[[1]]
28

The following statements create an array and a row, and populates them with data:

```
CREATE OR REPLACE VARIABLE x1 ARRAY (10) OF INT;
CREATE OR REPLACE VARIABLE x2 ROW( a1 INT, b1 ARRAY(10) OF INT );
SELECT ARRAY_AGG( "id" ) INTO x1 FROM GROUPO.Products;
SET x2.a1 = 10;
SET x2.b1 = x1;
```

The following statement returns the data in the array and row together:

```
SELECT x2.a1 AS a1, X, Z FROM UNNEST( x2.b1 ) WITH ORDINALITY AS DT(X,Z);
```

a1	X	Z
10	300	1
10	301	2
10	302	3
10	400	4
10	401	5
10	500	6
10	501	7
10	600	8
10	601	9

a1	X	Z
10	700	10

The following statement returns the first values in the array and row:

```
SELECT x2.a1, x2.b1[[1]];
```

a1	x2.b1[[1]]
10	300

The following statements create an array of ROW and populate it with data

```
CREATE OR REPLACE VARIABLE x4 ARRAY(10) OF ROW( a1 INT, b1 INT, c1
VARCHAR(120) );
SELECT ARRAY_AGG( ROW( ID, Quantity, Name ) ) INTO x4 FROM GROUPO.Products;
```

The following statement returns an unnested row from the array:

```
SELECT (x).a1 FROM UNNEST( x4 ) AS dt(x);
```

expression
300
301
302
400
401
500
501
600
601
700

Related Information

ROW and ARRAY Composite Data Types [page 185]
Comparisons of Composite Types [page 198]
ARRAY Constructor [Composite] [page 243]

FROM Clause [page 1173]

1.1.5.6 Bitwise Operators

Several operators can be used on bit data types, integer data types (including all variants such as bit, tinyint, smallint and so on), binary values, and bit array data types.

Operator	Description
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
~	bitwise NOT

The bitwise operators &, | and ~ are not interchangeable with the logical operators AND, OR, and NOT.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement selects rows in which the correct bits are set. For example, if the value of Options is `0x1001` then the row would be included.

```
SELECT *
FROM tableA
WHERE ( Options & 0x0101 ) <> 0;
```

1.1.5.7 Join Operators

SQL Anywhere supports two additional comparison operators, *= and =*, which are the Transact-SQL outer join operators.

When one of these operators is used in a comparison predicate, an implicit LEFT or RIGHT OUTER JOIN is specified.

> **i** Note
>
> Support for Transact-SQL outer join operators *= and =* is deprecated. To use Transact-SQL outer joins, the tsql_outer_joins database option must be set to On.

Related Information

Transact-SQL Outer Joins (*= or =*)
tsql_outer_joins Option

1.1.5.8 Operator Precedence

The precedence of operators in expressions is significant.

The operators at the top of the following list are evaluated before those at the bottom of the list.

1. unary operators (operators that require a single operand)
2. &, |, ^, ~
3. *, /, %
4. +, -
5. ||
6. **not**
7. **and**
8. **or**

When you use more than one operator in an expression, make the order of operation explicit using parentheses.

1.1.6 Expressions in SQL Statements

An expression is a statement that can be evaluated to return values.

Syntax

```
expression:
 case-expression
| constant
| [correlation-name.]column-name
| - expression
| expression operator expression
| ( expression )
| function-name ( expression, ... )
| if-expression
| special value
| ( subquery )
| variable-name
| sequence-expression
```

```
case-expression :
CASE expression
WHEN expression
THEN expression,...
[ ELSE expression ]
END
```

```
alternative form of case-expression :
CASE
WHEN search-condition
THEN expression, ...
[ ELSE expression ]
END
```

```
constant :
 integer | number | string | host-variable
```

```
special-value :
 CURRENT { DATE |  TIME |  TIMESTAMP }
| NULL
| SQLCODE
| SQLSTATE
| USER
```

```
if-expression :
IF condition
THEN expression
[ ELSE expression ]
ENDIF
```

```
sequence-expression :
sequence-name .[ CURRVAL | NEXTVAL ]
```

```
java-ref:
 .field-name [ java-ref ]
| >> field-name [ java-ref ]
| .method-name ( [ expression,... ] ) [ java-ref ]
| >> method-name ( [ expression,...] ) [ java-ref ]
```

```
operator:
{  + |  - |  * |  / |  || |  % }
```

Remarks

Expressions are used in many different places.

Expressions are formed from several different kinds of elements. These are discussed in the sections on functions and variables.

You must be connected to the database to evaluate expressions.

Side Effects

None.

In this section:

Constants in Expressions [page 37]
: Constants are numbers or string literals. String constants are enclosed in apostrophes ('single quotes'). An apostrophe is represented inside a string by two apostrophes in a row.

Column Names in Expressions [page 37]
: A column name is an identifier preceded by an optional correlation name. A correlation name is usually a table name.

Subqueries in Expressions [page 37]
: A subquery is a SELECT statement that is nested inside another SELECT, INSERT, UPDATE, or DELETE statement, or another subquery.

IF Expressions [page 38]
: An IF expression tests whether a condition is TRUE, FALSE, or UNKNOWN.

CASE Expressions [page 39]
: The CASE expression provides conditional SQL expressions. Case expressions can be used anywhere an expression can be used.

Regular Expressions Overview [page 40]
: A **regular expression** is a sequence of characters, wildcards, or operators that defines a pattern to search for within a string.

Regular Expressions Syntax [page 41]
: Regular expressions are supported with the SIMILAR TO, and REGEXP search conditions, and the REGEXP_SUBSTR function.

Regular Expressions Examples [page 54]
: There are many helpful examples of regular expressions you can refer to.

Compatibility of Expressions [page 56]
: SQL Anywhere uses the ANSI/ISO SQL Standard convention whereby strings enclosed in apostrophes are constant expressions, and strings enclosed in quotation marks (double quotes) are delimited identifiers (names for database objects).

Related Information

Special Values [page 87]
SQL Functions [page 206]
SQL Variables [page 123]
Search Conditions [page 58]
SQL Data Types [page 129]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.1.6.1 Constants in Expressions

Constants are numbers or string literals. String constants are enclosed in apostrophes ('single quotes'). An apostrophe is represented inside a string by two apostrophes in a row.

1.1.6.2 Column Names in Expressions

A column name is an identifier preceded by an optional correlation name. A correlation name is usually a table name.

If a column name has characters other than letters, digits and underscore, it must be surrounded by quotation marks (""). For example, the following are valid column names:

- Employees.Name
- address
- "date hired"
- "salary"."date paid"

Related Information

Identifiers [page 6]
FROM Clause [page 1173]

1.1.6.3 Subqueries in Expressions

A subquery is a SELECT statement that is nested inside another SELECT, INSERT, UPDATE, or DELETE statement, or another subquery.

If a subquery matches no rows, it evaluates to NULL.

The SELECT statement must be enclosed in parentheses, and must contain one and only one SELECT list item. When used as an expression, a subquery is generally allowed to return only one value.

A subquery can be used anywhere that a column name can be used. For example, a subquery can be used in the SELECT list of another SELECT statement.

Related Information

Subqueries in Search Conditions [page 61]

1.1.6.4 IF Expressions

An IF expression tests whether a condition is TRUE, FALSE, or UNKNOWN.

This is the syntax of the IF expression:

```
IF condition
THEN expression1
[ ELSE expression2 ]
{ ENDIF | END IF }
```

This expression returns the following:

- If `condition` evaluates to TRUE, the IF expression returns `expression1`.
- If `condition` evaluates to FALSE, the IF expression returns `expression2`.
- If `condition` evaluates to FALSE, and there is no `expression2`, the IF expression returns NULL.
- If `condition` evaluates to UNKNOWN, the IF expression returns NULL.

`expression1` is evaluated only if `condition` is TRUE. Similarly, `expression2` is evaluated only if `condition` is FALSE. Both `expression1` and `expression2` are arbitrary expressions; `condition` is any valid search condition.

> **i** Note
>
> The IF expression is not the same as the IF statement.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

IF Statement [page 1220]
Search Conditions [page 58]
NULL Special Value [page 103]

1.1.6.5 CASE Expressions

The CASE expression provides conditional SQL expressions. Case expressions can be used anywhere an expression can be used.

The syntax of the CASE expression is as follows:

```
CASE expression-1
WHEN expression-2
THEN expression-3, ...
[ ELSE expression-4 ]
{ END | END CASE }
```

If the expression following the CASE clause is equal to the expression following the WHEN clause, then the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

the CASE expression returns NULL if the ELSE clause doesn't exist and `expression-1`' doesn't match any of the `expression-2`...`expression-n` values.

For example, the following code uses a case expression as the second clause in a SELECT statement.

```
SELECT ID,
   ( CASE Name
      WHEN 'Tee Shirt' THEN 'Shirt'
      WHEN 'Sweatshirt' THEN 'Shirt'
      WHEN 'Baseball Cap' THEN 'Hat'
      ELSE 'Unknown'
   END ) AS Type
FROM GROUPO.Products;
```

An alternative syntax is as follows:

```
CASE
WHEN search-condition
THEN expression-1, ...
[ ELSE expression-2 ]
END [ CASE ]
```

If the search-condition following the WHEN clause is satisfied, the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

For example, the following statement uses a case expression as the third clause of a SELECT statement to associate a string with a search-condition.

```
SELECT ID, Name,
   ( CASE
      WHEN Name='Tee Shirt' THEN 'Sale'
      WHEN Quantity >= 50  THEN 'Big Sale'
      ELSE 'Regular price'
   END ) AS Type
FROM GROUPO.Products;
```

NULLIF Function for Abbreviated CASE Expressions

The NULLIF function provides a way to write some CASE clauses in short form. The syntax for NULLIF is as follows:

```
NULLIF ( expression-1, expression-2 )
```

NULLIF compares the values of the two expressions. If the first expression equals the second expression, NULLIF returns NULL. If the first expression does not equal the second expression, NULLIF returns the first expression.

> **i** Note
>
> Do not confuse the syntax of the CASE expression with that of the CASE clause.

Standards

ANSI/ISO SQL Standard

Core Feature. The standard permits any expression referenced by the statement to be evaluated at any point during execution. In the software, expression evaluation occurs when each WHEN clause is evaluated, in their syntactic order, with the exception of constant values that can be determined at compile time.

Support for END CASE with CASE expressions, in addition to END, is not in the standard. The standard defines END for use with CASE expressions and END CASE for use with CASE clauses.

Related Information

CASE Statement [page 800]

1.1.6.6 Regular Expressions Overview

A **regular expression** is a sequence of characters, wildcards, or operators that defines a pattern to search for within a string.

Regular expressions are supported as part of a REGEXP or SIMILAR TO search conditions in the WHERE clause of a SELECT statement, or as an argument to the REGEXP_SUBSTR function. The LIKE search condition does not support regular expressions, although some of the wildcards and operators you can specify with LIKE resemble the regular expression wildcards and operators.

The following SELECT statement uses a regular expression (`(K|C[^h])%`) to search the Contacts table and return contacts whose last name begins with K or C, but not Ch:

```
SELECT Surname, GivenName
   FROM GROUPO.Contacts
```

```
    WHERE Surname SIMILAR TO '(K|C[^h])%';
```

A regular expression can include additional syntax to specify grouping, quantification, assertions, and alternation.

Grouping

Grouping allows you to group parts of a regular expression to apply some additional matching criteria. For example, `'(abc){2}'` matches abcabc.

You can also use grouping to control the order in which the parts of the expression are evaluated. For example, `'ab(cdcd)'` looks first for an incidence of cdcd, and then evaluates whether the instance of cdcd is preceded by ab.

Quantification

Quantification allows you to control the number of times the preceding part of the expression can occur. For example, a question mark (?) is a quantifier that matches zero or one instance of the previous character. So, `'honou?r'` matches both honor and honour.

Assertions

Normally, searching for a pattern returns that pattern. Assertions allow you to test for the presence of a pattern, without having that pattern become part of what is returned. For example, `'SQL(?= Anywhere)'` matches SQL only if it is followed by a space and then Anywhere.

Alternation

Alternation allows you to specify alternative patterns to search for if the preceding pattern cannot be found. Alternate patterns are evaluated from left to right, and searching stops at the first match. For example, `'col(o|ou)r'` looks for an instance of color. If no instance is found, colour is searched for instead.

Related Information

Regular Expressions Syntax [page 41]
LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]
Search Conditions [page 58]
REGEXP Search Condition [page 72]
SIMILAR TO Search Condition [page 74]
REGEXP_SUBSTR Function [String] [page 506]

1.1.6.7 Regular Expressions Syntax

Regular expressions are supported with the SIMILAR TO, and REGEXP search conditions, and the REGEXP_SUBSTR function.

For SIMILAR TO, regular expression syntax is consistent with the ANSI/ISO SQL standard. For REGEXP and REGEXP_SUBSTR, regular expression syntax and support is consistent with Perl 5.

Regular expressions are used by REGEXP and SIMILAR TO to match a *string*, whereas regular expressions are used by REGEXP_SUBSTR to match a *substring*. To achieve substring matching behavior for REGEXP and

SIMILAR TO, you can specify wildcards on either side of the pattern you are trying to match. For example, `REGEXP '.*car.*'` matches car, carwash, and vicar. Or, you can rewrite your query to use the REGEXP_SUBSTR function.

Regular expression matching with SIMILAR TO is case- and accent-insensitive. REGEXP and REGEXP_SUBSTR is not affected by the database accent and case sensitivity.

In this section:

1.1.6.7.1 Regular Expressions: Metacharacters

Metacharacters are symbols or characters that have a special meaning within a regular expression.

The treatment of metacharacters can vary depending on:

- whether the regular expression is being used with the SIMILAR TO or REGEXP search conditions, or the REGEXP_SUBSTR function
- whether the metacharacter is inside of a character class in the regular expression

Before continuing, you should understand the definition of a **character class**. A character class is a set of characters enclosed in square brackets, against which characters in a string are matched. For example, in the syntax `SIMILAR TO 'ab[1-9]'`, [1-9] is a character class and matches one digit in the range of 1 to 9, inclusive. The treatment of metacharacters in a regular expression can vary depending on whether the metacharacter is placed inside a character class. Specifically, most metacharacters are handled as regular characters when positioned inside of a character class.

For SIMILAR TO (only), the metacharacters *, ?, +, _, |, (,), { must be escaped within a character class.

To include a literal minus sign (-), caret (^), or right-angle bracket (]) character in a character class, it must be escaped.

The list of supported regular expression metacharacters is provided below. Almost all metacharacters are treated the same when used by SIMILAR TO, REGEXP, and REGEXP_SUBSTR:

Character	Additional information
[ ]	Left and right square brackets are used to specify a **character class**. A character class is a set of characters to match against. With the exception of the hyphen (-) and the caret (^), metacharacters and quantifiers (such as * and {m}, respectively) specified within a character class have no special meaning and are evaluated as actual characters. Sub-character classes such as POSIX character classes are also supported.
*	The asterisk can be used to match a character 0 or more times. For example, `REGEXP '.*abc'` matches a string that ends with abc, and starts with any prefix. So, aabc, xyzabc, and abc match, but bc and abcc do not.
?	The question mark can be used to match a character 0 or 1 times. For example, `'colou?r'` matches color and colour.
+	The plus sign can be used to match a character 1 or more times. For example, `'bre+'` matches bre and bree, but not br.
-	A hyphen can be used within a character class to denote a range. For example, `REGEXP '[a-e]'` matches a, b, c, d, and e.
%	The percent sign can be used with SIMILAR TO to match any number of characters. The percent sign is not considered a metacharacter for REGEXP and REGEXP_SUBSTR. When specified, it matches a percent sign (%).
_	The underscore can be used with SIMILAR TO to match a single character. The underscore is not considered a metacharacter for REGEXP and REGEXP_SUBSTR. When specified, it matches an underscore (_).
|	The pipe symbol is used to specify alternative patterns to use for matching the string. In a string of patterns separated by a vertical bar, the vertical bar is interpreted as an OR and matching stops at the first match made starting from the leftmost pattern. So, you should list the patterns in descending order of preference. You can specify an unlimited number of alternative patterns.

Character	Additional information
`()`	Left and right parenthesis are metacharacters when used for grouping parts of the regular expression. For example, `(ab)*` matches zero or more repetitions of ab. As with mathematical expressions, you use grouping to control the order in which the parts of a regular expression are evaluated.
`{}`	Left and right curly braces are metacharacters when used for specifying **quantifiers**. Quantifiers specify the number of times a pattern must repeat to constitute a match. For example: **{m}** Matches a character exactly `m` times. For example, `'519-[0-9]{3}-[0-9]{4}'` matches a phone number in the 519 area code (providing the data is formatted in the manner defined in the syntax). **{m,}** Matches a character at least `m` times. For example, `'[0-9]{5,}'` matches any string of five or more digits. **{m,n}** Matches a character at least `m` times, but not more than `n` times. For example, `SIMILAR TO '_{5,10}'` matches any string with between 5 and 10 (inclusive) characters.
`\`	The backslash is used as an escape character for metacharacters. It can also be used to escape non-metacharacters.

Character	Additional information
^	For REGEXP and REGEXP_SUBSTR, when a caret is outside a character class, the caret matches the start of a string. For example, `'^[hc]at'` matches hat and cat, but only at the beginning of the string. When used inside a character class, the following behavior applies: **REGEXP and REGEXP_SUBSTR** When the caret is the first character in a character class, it matches anything other than the characters in the character set. For example, `REGEXP '[^abc]'` matches any character other than a, b, or c. If the caret is not the first character inside the square brackets, it matches a caret. For example, `REGEXP_SUBSTR '[a-e^c]'` matches a, b, c, d, e, and ^. **SIMILAR TO** For SIMILAR TO, the caret is treated as a subtraction operator. For example, `SIMILAR TO '[a-e^c]'` matches a, b, d, and e.
$	When used with REGEXP and REGEXP_SUBSTR, matches the end of a string. For example, `REGEXP 'cat$'` matches cat, but not catfish.
.	When used with REGEXP and REGEXP_SUBSTR, matches any single character. For example, `REGEXP 'a.cd'` matches any string of four characters that starts with a and ends with cd. When used with SIMILAR TO, matches a period (.).
:	The colon is used within a character set to specify a sub-character class. For example, `'[[:alnum:]]'`.

Related Information

LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]
Regular Expressions Examples [page 54]
REGEXP_SUBSTR Function [String] [page 506]
Regular Expressions: Special Sub-character Classes [page 46]

1.1.6.7.2 Regular Expressions: Special Sub-character Classes

Sub-character classes are special character classes embedded within a larger character class.

In addition to custom character classes where you define the set of characters to match (for example, `[abxq4]` limits the set of matching characters to a, b, x, q, and 4), SQL Anywhere supports sub-character classes such as most of the POSIX character classes. For example, `[[:alpha:]]` represents the set of all upper- and lowercase letters.

The REGEXP search condition and the REGEXP_SUBSTR function support all the syntax conventions in the table below, but the SIMILAR TO search expression does not. Conventions supported by SIMILAR TO have a Y in the SIMILAR TO column.

In REGEXP and when using the REGEXP_SUBSTR function, sub-character classes can be negated using a caret. For example, `[[:^alpha:]]` matches the set of all characters except alpha characters.

Sub-Character Class	Additional Information	SIMILAR TO
[:alpha:]	Matches upper- and lowercase alphabetic characters in the current collation. For example, `'[0-9]{3}[[:alpha:]]{2}'` matches three digits, followed by two letters.	Y
[:alnum:]	Match digits, and upper- and lowercase alphabetic characters in the current collation. For example, `'[[:alnum:]]+'` matches a string of one or more letters and numbers.	Y
[:digit:]	Match digits in the current collation. For example, `'[[:digit:]-]+'` matches a string of one or more digits or dashes. Likewise, `'[^[:digit:]-]+'` matches a string of one or more characters that are not digits or dashes.	Y
[:lower:]	Match lowercase alphabetic characters in the current collation. For example, `'[[:lower:]]'` does not match A because A is uppercase.	Y

Sub-Character Class	Additional Information	SIMILAR TO
[:space:]	Match a single blank (' '). For example, the following statement searches Contacts.City for any city with a two word name: `SELECT City FROM GROUPO.Contacts WHERE City REGEXP '.*[[:space:]].*';`	Y
[:upper:]	Match uppercase alphabetic characters in the current collation. For example, `'[[:upper:]ab]'` matches one of: any uppercase letter, a, or b.	Y
[:whitespace:]	Match a whitespace character such as space, tab, form feed, and carriage return.	Y
[:ascii:]	Match any seven-bit ASCII character (ordinal value between 0 and 127).	
[:blank:]	Match a blank space, or a horizontal tab. `[[:blank:]]` is equivalent to `[ \t]`.	
[:cntrl:]	Match ASCII characters with an ordinal value of less than 32, or character value 127 (control characters). Control characters include newline, form feed, backspace, and so on.	
[:graph:]	Match printed characters. `[[:graph:]]` is equivalent to `[[:alnum:][:punct:]]`.	
[:print:]	Match printed characters and spaces. `[[:print:]]` is equivalent to `[[:graph:][:whitespace:]]`.	

Sub-Character Class	Additional Information	SIMILAR TO
[:punct:]	Match one of: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~. The `[:punct:]` sub-character class may not include non-ASCII punctuation characters available in the current collation.	
[:word:]	Match alphabetic, digit, or underscore characters in the current collation. `[[:word:]]` is equivalent to `[[:alnum:]_]`.	
[:xdigit:]	Match a character that is in the character class [0-9A-Fa-f].	

Related Information

LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]
Regular Expressions Examples [page 54]
REGEXP_SUBSTR Function [String] [page 506]
Regular Expressions: Metacharacters [page 42]

1.1.6.7.3 Regular Expressions: Other Supported Syntax Conventions

The following syntax conventions are supported by the REGEXP search condition and the REGEXP_SUBSTR function, and they assume that the backslash is the escape character. *These conventions are not supported by the SIMILAR TO search expression.*

Regular Expression Syntax	Name and Meaning
\0 `xxx`	Matches the character whose value is \0`xxx`, where `xxx` is any sequence of octal digits, and 0 is a zero. For example, `\0134` matches a backslash.
\a	Matches the bell character.
\A	Used outside a character set to match the start of a string. Equivalent to ^ used outside a character set.

Regular Expression Syntax	Name and Meaning
\b	Matches a backspace character.
\B	Matches the backslash character (\).
\c x	Matches a named control character. For example, \cZ for ctrl-Z.
\d	Matches a digit in the current collation. For example, the following statement searches Contacts.Phone for all phone numbers that end with 00: `SELECT Surname, Surname, City, Phone FROM GROUPO.Contacts WHERE Phone REGEXP '\\d{8}00';` \d can be used both inside and outside character classes, and is equivalent to `[[:digit:]]`.
\D	Matches anything that is not a digit. This is the opposite of \d. \D can be used both inside and outside character classes, and is equivalent to `[^[:digit:]]`. Be careful when using the negated shorthands inside square brackets. `[\D\S]` is not the same as `[^\d\s]`. The latter matches any character that is not a digit or whitespace. So it matches x, but not 8. The former, however, matches any character that is either not a digit, or is not whitespace. Because a digit is not whitespace, and whitespace is not a digit, `[\D\S]` matches any character, digit, whitespace or otherwise.
\e	Matches the escape character.
\E	Ends the treatment of metacharacters as non-metacharacters, initiated by a \Q.
\f	Matches a form feed.
\n	Matches a new line.
\Q	Treat all metacharacters as non-metacharacters, until \E is encountered. For example, `\Q[$\E` is equivalent to `\[\$`.
\r	Matches a carriage return.

Regular Expression Syntax	Name and Meaning
\s	Matches a space or a character treated as whitespace. For example, the following statement returns all product names from Products.ProductName that have at least one space in the name: `SELECT Name FROM GROUPO.Products WHERE Name REGEXP '.*\\s.*'` \s can be used both inside and outside character classes, and is equivalent to `[[:whitespace:]]`.
\S	Matches a non-whitespace character. This is the opposite of \s, and is equivalent to `[^[:whitespace:]]`. \S can be used both inside and outside character classes. Be careful when using the negated shorthands inside square brackets. `[\D\S]` is not the same as `[^\d\s]`. The latter matches any character that is not a digit or whitespace. So it matches x, but not 8. The former, however, matches any character that is either not a digit, or is not whitespace. Because a digit is not whitespace, and whitespace is not a digit, `[\D\S]` matches any character, digit, whitespace or otherwise.
\t	Matches a horizontal tab.
\v	Matches a vertical tab.
\w	Matches an alphabetic character, digit, or underscore in the current collation. For example, the following statement returns all surnames from Contacts.Surname that are exactly seven alpha-numeric characters in length: `SELECT Surname FROM GROUPO.Contacts WHERE Surname REGEXP '\\w{7}';` \w can be used both inside and outside character classes. Equivalent to `[[:alnum:]_]`..
\W	Matches anything that is not an alphabetic character, digit, or underscore in the current collation. This is the opposite of \w, and is equivalent to `[^[:alnum:]_]`. This regular expression can be used both inside and outside character classes

Regular Expression Syntax	Name and Meaning
\x hh	Matches the character whose value is 0xhh, where hh is, at most, two hex digits. For example, \x2D is equivalent to a hyphen. Equivalent to \x{hh}.
\x{ hhh }	Matches the character whose value is 0xhhh, where hhh is, at most, eight hex digits.
\z	Matches the position (not character) at the end of the string. Equivalent to $.
\Z	Matches the position (not character) at the end of the string. Equivalent to $.

Related Information

LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]
Regular Expressions Examples [page 54]
REGEXP_SUBSTR Function [String] [page 506]
Regular Expressions: Metacharacters [page 42]
Regular Expressions: Special Sub-character Classes [page 46]

1.1.6.7.4 Regular Expressions: Assertions

Assertions test whether a condition is true, and affect the position in the string where matching begins. Assertions do not return characters; the assertion pattern is not included in the final match.

These assertions are supported by the REGEXP search condition and the REGEXP_SUBSTR function. These conventions are not supported by the SIMILAR TO search expression.

Lookahead and lookbehind assertions can be useful with REGEXP_SUBSTR when trying to split a string. For example, you can return the list of street names (without the street numbers) in the Address column of the Customers table by executing the following statement:

```
SELECT REGEXP_SUBSTR( Street, '(?<=^\\S+\\s+).*$' )
FROM GROUPO.Customers;
```

Another example is using a regular expression to verify that a password conforms to certain rules. You could use a zero width assertion similar to the following:

```
IF password REGEXP '(?=.*[[:digit:]])(?=.*[[:alpha:]].*[[:alpha:]])[[:word:]]
{4,12}' THEN
   MESSAGE 'Password conforms' TO CLIENT;
ELSE
```

```
   MESSAGE 'Password does not conform' TO CLIENT;
END IF
```

The password is valid when the following are true:

- `password` has at least one digit (zero width positive assertion with [[:digit:]])
- `password` has at least two alphabetic characters (zero width positive assertion with [[:alpha:]].*[[:alpha:]])
- `password` contains only alpha-numeric or underscore characters ([[:word:]])
- `password` is at least 4 characters, and at most 12 characters ({4,12})

The following table contains supported assertions:

Syntax	Meaning
(?= `pattern` **)**	**Positive lookahead zero-width assertion** Looks to see if the current position in the string is immediately followed by an occurrence of `pattern`, without `pattern` becoming part of the match string. `'A(?=B)'` matches an A that is followed by a B, without making the B part of the match. For example, `SELECT REGEXP_SUBSTR( 'in new york city', 'new(?=\\syork)');` returns the substring 'new' since it is immediately followed by ' york' (note the space before york).
(?! `pattern` **)**	**Negative lookahead zero-width assertions** Looks to see if the current position in the string is *not* immediately followed by an occurrence of `pattern`, without `pattern` becoming part of the match string. So, `'A(?!B)'` matches an A that is not followed by a B. For example,`SELECT REGEXP_SUBSTR('new jersey', 'new(?!\\syork)');` returns the substring new.
(?<= `pattern` **)**	**Positive lookbehind zero-width assertions** Looks to see if the current position in the string is immediately preceded by an occurrence of `pattern`, without `pattern` becoming part of the match string. So, `'(?<=A)B'` matches a B that is immediately preceded by an A, without making A part of the match. For example, `SELECT REGEXP_SUBSTR('new york', '(?<=new\\s)york');` returns the substring york.

Syntax	Meaning
`(?<! pattern )`	**Negative lookbehind zero-width assertions** Looks to see if the current position in the string is *not* immediately preceded by an occurrence of `pattern`, without `pattern` becoming part of the match string. For example, `SELECT REGEXP_SUBSTR('about york', '(?<!new\\s)york');` returns the sub-string york.
`(?> pattern )`	**Possessive local subexpression** Matches only the largest prefix of the remaining string that matches `pattern`. For example, in `'aa' REGEXP '(?>a*)a'`, `(?>a*)` matches (and consumes) the aa, and never just the leading a. As a result, `'aa' REGEXP '(?>a*)a'` evaluates to false.
`(?: pattern )`	**Non-capturing block** This is functionally equivalent to just `pattern`, and is provided for compatibility. For example, in `'bb' REGEXP '(?:b*)b'`, `(?:b*)` matches (and consumes) the bb. However, unlike possessive local subexpression, the last b in bb is given up to allow the whole match to succeed (that is, to allow the matching to the b found outside the non-capturing block). Likewise, `'a(?:bc|b)c'` matches abcc, and abc. In matching abc, backtracking on the final c in bc takes place so that the c outside the group can be used to make the match successful.
`(?# text )`	Used for comments. The content of `text` is ignored.

Related Information

LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]
Regular Expressions Examples [page 54]
REGEXP_SUBSTR Function [String] [page 506]
Regular Expressions: Metacharacters [page 42]
Regular Expressions: Special Sub-character Classes [page 46]

1.1.6.8 Regular Expressions Examples

There are many helpful examples of regular expressions you can refer to.

All examples work for REGEXP and some also work for SIMILAR TO, as noted in the Example column. Results vary depending on the search condition you use for searching. For those that work with SIMILAR TO, results can vary further depending on case and accent sensitivity.

Backslashes should be doubled if the examples are used in literal strings (for example, `'.+@.+\\..+'`)

Example	Sample Matches

Credit Card Numbers (REGEXP only):
Visa:
`4[0-9]{3}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
MasterCard:
`5[0-9]{3}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
American Express:
`37[0-9]{2}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
Discover:
`6011\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
	Matches (Visa): 4123 6453 2222 1746
	Non-Matches (Visa):
	3124 5675 4400 4567, 4123-6453-2222-1746
	Similarly, MasterCard matches a set of 16 numbers, starting with 5, with a space between each subset of four numbers. American Express and Discover are the same, but must start with 37 and 6011 respectively.

Dates (REGEXP and SIMILAR TO): `([0-2][0-9]|30|31)/(0[1-9]|1[0-2])/[0-9]{4}`	Matches: 31/04/1999, 15/12/4567 Non-Matches: 31/4/1999, 31/4/99, 1999/04/19, 42/67/25456
Windows absolute paths (REGEXP only): `([A-Za-z]:|\\\\)\\\\[[:alnum:][:whitespace:].!"#$%&''()+,.\\\\;=@\[^_`{}~\\]-]*`	Matches: \\server\share\file Non-Matches: \directory\directory2, /directory2
Email Addresses (REGEXP only): `[[:word:]\-.]+@[[:word:]\-.]+\.[[:alpha:]]{2,}`	Matches: abc.123@def456.com, _123@abc.ca Non-Matches: abc@dummy, ab*cd@efg.hijkl
Email Addresses (REGEXP only): `.+@.+\..+`	Matches: *@qrstuv@wxyz.12345.com, __1234^%@@abc.def.ghijkl Non-Matches: abc.123.*&ca, ^%abcdefg123

Example	Sample Matches
HTML Hexadecimal Color Codes (REGEXP and SIMILAR TO): `[A-F0-9]{6}`	Matches: AB1234, CCCCCC, 12AF3B Non-Matches: 123G45, 12-44-CC
HTML Hexadecimal Color Codes (REGEXP only): `[A-F0-9]{2}\s[A-F0-9]{2}\s[A-F0-9]{2}`	Matches: AB 11 00, CC 12 D3 Non-Matches: SS AB CD, AA BB CC DD, 1223AB
IP Addresses (REGEXP only):	Matches: 10.25.101.216 Non-Matches: 0.0.0, 256.89.457.02
Java Comments (REGEXP only): `/\*.*\*/|//[^\n]*((2(5[0-5]|[0-4][0-9])|1([0-9][0-9])|([1-9][0-9])|[0-9])\.){3}(2(5[0-5]|[0-4][0-9])|1([0-9][0-9])|([1-9][0-9])|[0-9])`	Matches Java comments that are between /* and */, or one line comments prefaced by //. Non-Matches: a=1
Money (REGEXP only): `(\+|-)?\$[0-9]*\.[0-9]{2}`	Matches: $1.00, -$97.65 Non-Matches: $1, 1.00$, $-75.17
Positive, negative numbers, and decimal values (REGEXP only): `(\+|-)?[0-9]+(\.[0-9]+)?`	Matches: +41, -412, 2, 7968412, 41, +41.1, -3.141592653 Non-Matches: ++41, 41.1.19, -+97.14
Passwords (REGEXP and SIMILAR TO): `((2(5[0-5[[:alnum:]]{4,10}`	Matches: abcd, 1234, A1b2C3d4, 1a2B3 Non-Matches: abc, *ab12, abcdefghijkl
Passwords (REGEXP only): `[a-zA-Z]\w{3,7}`	Matches: AB_cd, A1_b2c3, a123_ Non-Matches: *&^g, abc, 1bcd
Phone Numbers (REGEXP and SIMILAR TO): `([2-9][0-9]{2}-[2-9][0-9]{2}-[0-9]{4})|([2-9][0-9]{2}\s[2-9][0-9]{2}\s[0-9]{4})`	Matches: 519-883-6898, 519 888 6898 Non-Matches: 888 6898, 5198886898, 519 883-6898
Sentences (REGEXP only): `[A-Z0-9].*(\.|\?|!)`	Matches: Hello, how are you? Non-Matches: i am fine

Example	Sample Matches
Sentences (REGEXP only): `[[:upper:]0-9].*[.?!]`	Matches: Hello, how are you? Non-Matches: i am fine
Social Security Numbers (REGEXP and SIMILAR TO): `[0-9]{3}-[0-9]{2}-[0-9]{4}`	Matches: 123-45-6789 Non-Matches:123 45 6789, 123456789, 1234-56-7891
URLs (REGEXP only): `(http://)?www\.[a-zA-Z0-9]+(.co)?\.[a-zA-Z]{2,4}`	Matches: http://www.sample.com, www.sample.info, www.sample.co.jp Non-Matches: http://sample.com, http://www.sample.common

Related Information

Regular Expressions Syntax [page 41]
LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]

1.1.6.9 Compatibility of Expressions

SQL Anywhere uses the ANSI/ISO SQL Standard convention whereby strings enclosed in apostrophes are constant expressions, and strings enclosed in quotation marks (double quotes) are delimited identifiers (names for database objects).

In this section:

1.1.6.9.1 The quoted_identifier Option

You can use the quoted_identifier option to control the interpretation of delimited strings. By default, the quoted_identifier option is set to On.

Setting the Option

The following statement changes the setting of the quoted_identifier option to On:

```
SET quoted_identifier On;
```

The following statement changes the setting of the quoted_identifier option to Off:

```
SET quoted_identifier Off;
```

You cannot use quotation marks for identifiers that are SQL reserved words if the quoted_identifier option is Off. In the following example, the result set consists of many rows with the same column value 'option' when quoted_identifier is Off.

```
SELECT "option" FROM SYS.SYSOPTION;
```

However, square brackets or back quotes (ticks) can be used in place of quotation marks when quoted_identifier is Off. The following examples return the names of the options in the SYSOPTION system view.

```
SELECT [option] FROM SYS.SYSOPTION;
SELECT `option` FROM SYS.SYSOPTION;
```

Either of these latter two quoting methods is preferred over quotation marks for identifiers that are keywords because they are impervious to the setting of the quoted_identifier option.

Compatible Interpretation of Delimited Strings

You can choose to use either the ANSI/ISO SQL Standard or the default Transact-SQL convention as long as the quoted_identifier option is set to the same value in each DBMS.

Example

If you choose to operate with the quoted_identifier option On (the default setting), then the following statements involving the SQL keyword **user** are valid for both DBMSs.

```
CREATE TABLE "user" ( col1 char(5) )
go
INSERT "user" ( col1 ) VALUES ( 'abcde' )
go
```

If you choose to operate with the quoted_identifier option off then the following statement is valid for both DBMSs. In the following example, Chin is a string and not an identifier.

```
SELECT *
FROM GROUPO.Employees
WHERE Surname = "Chin"
go
```

Related Information

Reserved Words [page 6]
quoted_identifier Option

1.1.7 Search Conditions

A search condition is the criteria specified for a WHERE clause, a HAVING clause, a CHECK clause, an ON phrase in a join, or an IF expression. A search condition is also called a predicate.

Syntax

```
search-condition :
expression comparison-operator expression
| expression comparison-operator { [ ANY | SOME ] | ALL } ( subquery )
| expression IS [ NOT ] DISTINCT FROM expression
| expression IS [ NOT ] NULL
| expression [ NOT ] BETWEEN expression AND expression
| expression [ NOT ] LIKE pattern [ ESCAPE expression ]
| expression [ NOT ] SIMILAR TO pattern [ ESCAPE escape-expression ]
| expression [ NOT ] REGEXP pattern [ ESCAPE escape-expression ]
| expression [ NOT ] IN ( expression , ... )
    | ( query-expression )
| NOT search-condition
| CONTAINS (column-name [,... ] , query-string )
| EXISTS ( query-expression )
| search-condition [ { AND | OR } search-condition ] [ ... ]
| ( search-condition )
| ( search-condition , estimate )
|  search-condition IS [ NOT ] { TRUE | FALSE | UNKNOWN }
| expression IS [ NOT ] OF ( type-name [ ONLY ],... )
| trigger-operation
```

```
comparison-operator :
=
| >
| <
| >=
| <=
| <>
| !=
| !<
| !>
```

```
trigger-operation :
```

```
INSERTING
| DELETING
| UPDATING [ ( column-name-string ) ]
| UPDATE( column-name )
```

Parameters

- ALL search condition
- ANY and SOME search conditions
- IS [NOT] DISTINCT FROM search condition
- BETWEEN search condition
- CONTAINS search condition
- EXISTS search condition
- LIKE search condition
- SIMILAR TO search condition
- REGEXP search condition
- IS OF `type-expression`, and IS NOT OF `type-expression`
  This type predicate was added for support of spatial geometries, but it can be used for any existing data type as well.

Remarks

Search conditions are used either to choose a subset of the rows from a table, or in a control statement such as an IF statement to determine control of flow.

In SQL, every condition evaluates as one of TRUE, FALSE, or UNKNOWN. This is called three-valued logic. The result of a comparison is UNKNOWN if either value being compared is the NULL value.

Rows satisfy a search condition only if the result of the condition is TRUE. Rows for which the condition is UNKNOWN or FALSE do not satisfy the search condition.

Subqueries form an important class of expression that is used in many search conditions.

The LIKE, SIMILAR TO, and REGEXP search conditions are very similar.

Prerequisites

Must be connected to the database.

Side Effects

None.

In this section:

Subqueries in Search Conditions [page 61]
Subqueries that return exactly one column and either zero or one row can be used in any SQL statement wherever a column name could be used, including in the middle of an expression.

ALL Search Condition [page 61]
Use the ALL search condition to compare a value to all values in a set.

ANY and SOME Search Conditions [page 62]
Use the ANY or SOME search condition to compare a value to any value in a set.

IS DISTINCT FROM and IS NOT DISTINCT FROM Search Conditions [page 63]
Use the IS DISTINCT FROM and IS NOT DISTINCT FROM search conditions to evaluate whether a value is distinct from values in a set.

BETWEEN Search Condition [page 64]
Use the BETWEEN search condition to evaluate whether a value is between values in another set.

LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]
The REGEXP, LIKE, and SIMILAR TO search conditions are similar in that they all attempt to match a pattern to a string. Also, all three attempt to match an entire string, not a substring within the string.

IN Search Condition [page 75]
Use the IN search condition to evaluate whether a value is found in a set.

CONTAINS Search Condition [page 76]
Use the CONTAINS search condition to evaluate whether a value is contained in a set.

EXISTS Search Condition [page 83]
Use the EXISTS search condition to evaluate whether a value is found in a set.

IS NULL and IS NOT NULL Search Conditions [page 84]
Use the IS NULL search condition to evaluate whether a value in a set is NULL.

Truth Value Search Conditions [page 84]
Use the IS TRUE search condition to evaluate whether a condition evaluates to a specified value.

Three-valued Logic [page 85]
The following tables display how the AND, OR, NOT, and IS logical operators of SQL work in three-valued logic.

Explicit Selectivity Estimates [page 86]
The database server uses statistical information to determine the most efficient strategy for executing each statement.

Related Information

Spatial Data Type Syntax
Expressions in SQL Statements [page 34]

LIKE Search Condition [page 67]
SIMILAR TO Search Condition [page 74]
REGEXP Search Condition [page 72]
NULL Special Value [page 103]

1.1.7.1 Subqueries in Search Conditions

Subqueries that return exactly one column and either zero or one row can be used in any SQL statement wherever a column name could be used, including in the middle of an expression.

For example, expressions can be compared to subqueries in comparison conditions as long as the subquery does not return more than one row. If the subquery (which must have exactly one column) returns one row, then the value of that row is compared to the expression. If a subquery returns no rows, the value of the subquery is NULL.

Subqueries that return exactly one column and any number of rows can be used in IN, ANY, ALL, and SOME search conditions. Subqueries that return any number of columns and rows can be used in EXISTS search conditions.

Standards

ANSI/ISO SQL Standard

Core Feature.

Related Information

Comparison Operators [page 15]

1.1.7.2 ALL Search Condition

Use the ALL search condition to compare a value to all values in a set.

Syntax

```
expression comparison-operator ALL ( subquery )
```

```
comparison-operator:
=
| >
| <
| >=
| <=
```

```
| <>
| !=
| !<
| !>
```

Remarks

With the ALL search condition, if the value of subquery result set is the empty set, the search condition evaluates to TRUE. Otherwise, the search condition evaluates to TRUE, FALSE, or UNKNOWN, depending on the value of `expression`, and the result set returned by the subquery, as follows:

If the expression value is...	and the result set returned by the subquery contains at least one NULL, then...	or the result set returned by the subquery contains no NULLs, then...
NULL	UNKNOWN	UNKNOWN
not NULL	If there exists at least one value in the subquery result set for which the comparison with the expression value is FALSE, then the search condition evaluates to FALSE. Otherwise, the search condition evaluates to UNKNOWN.	If there exists at least one value in the subquery result set for which the comparison with the expression value is FALSE, then the search condition evaluates to FALSE. Otherwise, the search condition evaluates to TRUE.

Standards

ANSI/ISO SQL Standard

Core Feature.

1.1.7.3 ANY and SOME Search Conditions

Use the ANY or SOME search condition to compare a value to any value in a set.

Syntax

```
expression comparison-operator { ANY | SOME }( subquery )
```

```
comparison-operator:
=
| >
| <
| >=
| <=
| <>
| !=
| !<
```

```
| />
```

Remarks

The keywords ANY and SOME are synonymous.

With the ANY and SOME search conditions, if the subquery result set is the empty set, the search condition evaluates to FALSE. Otherwise, the search condition evaluates to TRUE, FALSE, or UNKNOWN, depending on the value of `expression`, and the result set returned by the subquery, as follows:

If the expression value is...	and the result set returned by the subquery contains at least one NULL, then...	or the result set returned by the subquery contains no NULLs, then...
NULL	UNKNOWN	UNKNOWN
not NULL	If there exists at least one value in the subquery result set for which the comparison with the expression value is TRUE, then the search condition evaluates to TRUE. Otherwise, the search condition evaluates to UNKNOWN.	If there exists at least one value in the subquery result set for which the comparison with the expression value is TRUE, then the search condition evaluates to TRUE. Otherwise, the search condition evaluates to FALSE.

An ANY or SOME search condition with an equality operator, evaluates to TRUE if `expression` is equal to any of the values in the result of the subquery, and FALSE if the value of the expression is not NULL, does not equal any of the values in the result of the subquery, and the result set doesn't contain NULLs.

> **i** Note
>
> The usage of **= ANY** or **= SOME** is equivalent to using the IN keyword.

Standards

ANSI/ISO SQL Standard

Core Feature.

1.1.7.4 IS DISTINCT FROM and IS NOT DISTINCT FROM Search Conditions

Use the IS DISTINCT FROM and IS NOT DISTINCT FROM search conditions to evaluate whether a value is distinct from values in a set.

> Syntax

```
expression1 IS [ NOT ] DISTINCT FROM expression2
```

Remarks

The IS DISTINCT FROM and IS NOT DISTINCT FROM search conditions are sargable and evaluate to TRUE or FALSE.

The IS NOT DISTINCT FROM search condition evaluates to TRUE if `expression1` is equal to `expression2`, or if both expressions are NULL. This is equivalent to a combination of two search conditions, as follows:

```
(expression1 = expression2) IS TRUE OR ( expression1 IS NULL AND expression2 IS
NULL )
```

The IS DISTINCT FROM syntax reverses the meaning. That is, IS DISTINCT FROM evaluates to TRUE if `expression1` is not equal to `expression2`, and at least one of the expressions is not NULL. This is equivalent to the following:

```
NOT(( expression1 = expression2) IS TRUE OR ( expression1 IS NULL AND
expression2 IS NULL ))
```

Standards

ANSI/ISO SQL Standard

The IS [NOT] DISTINCT FROM predicate is defined in the ANSI/ISO SQL Standard. The IS DISTINCT FROM predicate is Feature T151, "DISTINCT predicate". The IS NOT DISTINCT FROM predicate is Feature T152, "DISTINCT predicate with negation".

Related Information

Query Predicates

1.1.7.5 BETWEEN Search Condition

Use the BETWEEN search condition to evaluate whether a value is between values in another set.

Syntax

```
expression [ NOT ] BETWEEN start-expression AND end-expression
```

Remarks

The BETWEEN search condition can evaluate as TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the search condition evaluates as TRUE if `expression` is between `start-expression` and `end-expression`. The NOT keyword reverses the meaning of the search condition but leaves UNKNOWN unchanged.

The BETWEEN search condition is equivalent to a combination of two inequalities:

```
[ NOT ] ( expression >= start-expression AND expression <= end-expression )
```

Standards

ANSI/ISO SQL Standard

Core Feature.

1.1.7.6 LIKE, REGEXP, and SIMILAR TO Search Conditions

The REGEXP, LIKE, and SIMILAR TO search conditions are similar in that they all attempt to match a pattern to a string. Also, all three attempt to match an entire string, not a substring within the string.

The basic syntax for all three search conditions is similar:

```
expression search-condition pattern
```

LIKE, REGEXP, and SIMILAR TO: Differences in `pattern` Definition

- REGEXP supports a superset of regular expression syntax supported by SIMILAR TO. In addition, for compatibility with other products, the REGEXP search condition supports several syntax extensions. Also, REGEXP and SIMILAR TO have a different default escape character and process the characters underscore (_), percent (%), and caret (^) differently. REGEXP behavior matches closely with Perl 5 (except where Perl syntax and operators are not supported).
- LIKE syntax for `pattern` is simple and supports a small set of wildcards, but does not support the full regular expression syntax.
- SIMILAR TO syntax for `pattern` allows a robust pattern matching using the regular expression syntax defined in the ANSI/ISO SQL standard.

LIKE, REGEXP, and SIMILAR TO: Differences in Character Comparisons

When performing comparisons, REGEXP behavior is different from LIKE and SIMILAR TO. For REGEXP comparisons, the database server uses code point values in the **database character set** for comparisons. This is consistent with other regular expression implementations such as Perl.

For LIKE and SIMILAR TO, the database server uses the equivalence and sort order in the **database collation** for comparisons. This is consistent with how the database evaluates comparison operators such as > and =.

The difference in character comparison methods means that results for matching and range evaluation for REGEXP and LIKE/SIMILAR differ as well.

Differences in matching

Since REGEXP uses code point values, it only matches a literal in a pattern if it is the exact same character. REGEXP matching is therefore not impacted by such things as database collation, case-sensitivity, or accent sensitivity. For example, 'A' could never be returned as a match for 'a'.

Since LIKE and SIMILAR TO use the database collation, results are impacted by case- and accent-sensitivity when determining character equivalence. For example, if the database collation is case- and accent-insensitive, matches are case- and accent-insensitive. So, an 'A' could be returned as a match for 'a'.

Differences in range evaluation

Since REGEXP uses code points for range evaluation, a character is considered to be in the range if its code point value is equal to, or between, the code point values for the start and end of the range. For example, the comparison `x REGEXP '[A-C]'`, for the single character `x`, is equivalent to `CAST(x AS BINARY) >= CAST(A AS BINARY) AND CAST(x AS BINARY) <= CAST(C AS BINARY)`.

Since LIKE and SIMILAR TO use the collation sort order for range evaluation, a character is considered to be in the range if its position in the collation is the same as, or between, the position of the start and end characters for the range. For example, the comparison `x SIMILAR TO '[A-C]'` (where `x` is a single character) is equivalent to `x >= A AND x <= C`, and the comparison operators are evaluated using the collation sort ordering.

The following table shows the set of characters included in the range `'[A-C]'` as evaluated by LIKE, SIMILAR TO, and REGEXP. Both databases use the 1252LATIN1 collation, but the first database is case-insensitive, while the second one is case sensitive.

	LIKE/SIMILAR TO '[A-C]'	REGEXP '[A-C]'
demo.db (case-insensitive)	A,B,C,a,b,c,ª,À,Á,Â,Ã,Ä,Å,Æ,Ç,à,á,â,ã,ä,å,æ,ç	A, B, C
`charsensitive.db` (case-sensitive)	A,B,C,b,c,À,Á,Â,Ã,Ä,Å,Æ,Ç,ç	A, B, C

The following can be observed in the results:

- LIKE and SIMILAR TO include accented characters in the range.
- LIKE and SIMILAR TO include different characters depending on database case-sensitivity. Specifically, they include any lowercase letters found within the range, which you may not have anticipated when searching on a case-sensitive database.

Similarly, on a case-sensitive database, some characters included in the range might appear to be inconsistent. For example, `SIMILAR TO '[A-C]'` on a case-sensitive database includes A, b, B, c, C but not a because a occurs before the uppercase A in the sort order.

- REGEXP returns only A, B, C regardless of database case sensitivity. If you want the range to include lowercase letters, you must add them to the range definition. For example, `REGEXP '[a-cA-C]'`.
- the REGEXP set of characters does not change, regardless of database case-sensitivity.

Even though your database uses a different collation, or has different case- or accent-sensitivity settings than the examples above, you can perform a similar test to see what is returned by LIKE, SIMILAR TO, or REGEXP by connecting to the database and executing any of these statements:

```
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) LIKE '[A-C]';
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) REGEXP '[A-C]';
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) SIMILAR TO '[A-
C]';
```

In this section:

LIKE Search Condition [page 67]
Use the LIKE search condition to evaluate whether a value is similar to values in a set.

REGEXP Search Condition [page 72]
Matches a pattern against a string.

SIMILAR TO Search Condition [page 74]
Matches a pattern against a string.

Related Information

Regular Expressions Overview [page 40]
Regular Expressions Syntax [page 41]
Regular Expressions Examples [page 54]

1.1.7.6.1 LIKE Search Condition

Use the LIKE search condition to evaluate whether a value is similar to values in a set.

Syntax

The syntax for the LIKE search condition is as follows:

```
expression [ NOT ] LIKE pattern [ ESCAPE escape-character ]
```

Parameters

expression

The string to be searched.

pattern

The pattern to search for within `expression`.

escape-character

The character to use to escape special characters such as underscores and percent signs.

Remarks

The LIKE search condition attempts to match `expression` with `pattern` and evaluates to TRUE, FALSE, or UNKNOWN.

The search condition evaluates to TRUE if `expression` matches `pattern` (assuming NOT was not specified). If either `expression` or `pattern` is the NULL value, the search condition evaluates to UNKNOWN. The NOT keyword reverses the meaning of the search condition, but leaves UNKNOWN unchanged.

`expression` is interpreted as a CHAR or NCHAR string. The entire contents of `expression` is used for matching. Similarly, `pattern` is interpreted as a CHAR or NCHAR string and can contain any number of the supported wildcards from the following table:

Wildcard	Matches
_ (underscore)	Any one character. For example, `a_` matches ab and ac, but not a.
% (percent)	Any string of zero or more characters. For example, `bl%` matches bl and bla.
[]	Any single character in the specified range or set. For example, `T[oi]m` matches Tom or Tim.
[^]	Any single character *not* in the specified range or set. For example, `M[^c]` matches Mb and Md, but not Mc.

All other characters must match exactly.

For example, the following search condition returns TRUE for any row where name starts with the letter a and has the letter b as its second last character.

```
... name LIKE 'a%b_'
```

If `escape-character` is specified, it must evaluate to a single-byte CHAR or NCHAR character. The escape character can precede a percent, an underscore, a left square bracket, or another escape character in the `pattern` to prevent the special character from having its special meaning. When escaped in this manner, a percent matches a percent, and an underscore matches an underscore.

Patterns up to 126 bytes in length are supported.

Different Ways to use the LIKE Search Condition

To search for	Example	Additional information
One of a set of characters	`LIKE 'sm[iy]th'`	A set of characters to look for is specified by listing the characters inside square brackets. In this example, the search condition matches *smith* and *smyth*.
One of a range of characters	`LIKE '[a-r]ough'`	A range of characters to look for is specified by giving the ends of the range inside square brackets, separated by a hyphen. In this example, the search condition matches bough and rough, but not tough. The range of characters [a-z] is interpreted as "greater than or equal to a, and less than or equal to z", where the greater than and less than operations are carried out within the collation of the database. The lower end of the range must precede the higher end of the range. For example, `[z-a]` does not match anything because no character matches the [z-a] range.
Ranges and sets combined	`... LIKE '[a-rt]ough'`	You can combine ranges and sets within square brackets. In this example, `... LIKE '[a-rt]ough'` matches bough, rough, and tough. The pattern [a-rt] is interpreted as exactly one character that is either in the range a to r inclusive, or is t.
One character not in a range	`... LIKE '[^a-r]ough'`	The caret character (^) is used to specify a range of characters that is excluded from a search. In this example, `LIKE '[^a-r]ough'` matches the string tough, but not the strings rough or bough. The caret negates the rest of the contents of the brackets. For example, the bracket [^a-rt] is interpreted as exactly one character that is not in the range a to r inclusive, and is not t.

To search for	Example	Additional information
Search patterns with trailing blanks	`'90 '`, `'90[ ]'` and `'90_'`	When your search pattern includes trailing blanks, the database server matches the pattern only to values that contain blanks. It does not blank pad strings. For example, the patterns '90 ', '90[ ]', and '90_' match the expression '90 ', but do not match the expression '90', even if the value being tested is in a CHAR or VARCHAR column that is three or more characters in width.

Special Cases of Ranges and Sets

Any single character in square brackets means that character. For example, `[a]` matches just the character a. `[^]` matches just the caret character, `[%]` matches just the percent character (the percent character does not act as a wildcard in this context), and `[_]` matches just the underscore character. Also, `[[]` matches just the character [.

Other special cases are as follows:

- The pattern `[a-]` matches either of the characters a or -.
- The pattern `[]` is never matched and always returns no rows.
- The patterns `[` or `[abp-q` return syntax errors because they are missing the closing bracket.
- You cannot use wildcards inside square brackets. The pattern `[a%b]` finds one of a, %, or b.
- You cannot use the caret character to negate ranges except as the first character in the bracket. The pattern `[a^b]` finds one of a, ^, or b.

Case Sensitivity and how Comparisons are Performed

If the database collation is case sensitive, the search condition is also case sensitive. To perform a case insensitive search with a case sensitive collation, you must include upper and lower characters. For example, the following search condition evaluates to true for the strings Bough, rough, and TOUGH:

```
LIKE '[a-zA-Z][oO][uU][gG][hH]'
```

Comparisons are performed character by character, unlike the equivalence (=) operator and other operators where the comparison is done string by string. For example, when a comparison is done in a UCA collation (CHAR or NCHAR with the collation set to UCA), `'Æ'='AE'` is true, but `'Æ' LIKE 'AE'` is false.

For a character-by-character comparison to match, each single character in the expression being searched must match a single character (using the collation's character equivalence), or a wildcard in the LIKE expression.

National Character (NCHAR) Support

LIKE search conditions can be used to compare CHAR and NCHAR strings. In this case, character set conversion is performed so that the comparison is done using a common data type. Then, a character-by-character comparison is performed.

You can specify `expression` or `pattern` as an NCHAR string literal by prefixing the quoted value with N (for example, `expression LIKE N'pattern'`). You can also use the CAST function to cast the pattern to CHAR or NCHAR (for example, `expression LIKE CAST(pattern AS datatype)`.

Blank Padded Databases

The semantics of a LIKE pattern does not change if the database is blank-padded since matching `expression` to `pattern` involves a character-by-character comparison in a left-to-right fashion. No additional blank padding is performed on the value of either `expression` or `pattern` during the evaluation. Therefore, the expression `a1` matches the pattern a1, but not the patterns 'a1 ' (a1, with a space after it) or a1_.

Standards

ANSI/ISO SQL Standard

The LIKE search condition is a core feature of the ANSI/ISO SQL Standard. However, there are subtle differences in behavior from that of the standard because the software supports case-insensitive collations and blank-padding.

The software supports optional ANSI/ISO SQL Language Feature F281, which permits the pattern and escape-expressions to be arbitrary expressions evaluated at execution time. Feature F281 also permits `expression` to be an expression more complex than a simple column reference.

The use of character ranges and sets contained in square brackets [] is not in the standard.

The software supports ANSI/ISO SQL Feature T042, which permits LIKE search conditions to reference string-expressions that are LONG VARCHAR values.

LIKE search conditions that specify NCHAR string expressions or patterns is optional ANSI/ISO SQL Language Feature F421.

Related Information

Comparisons Between CHAR and NCHAR [page 195]
String Literals [page 13]
The WHERE Clause: Specifying Rows
CAST Function [Data Type Conversion] [page 275]
REGEXP Search Condition [page 72]

SIMILAR TO Search Condition [page 74]

1.1.7.6.2 REGEXP Search Condition

Matches a pattern against a string.

Syntax

```
expression [ NOT ] REGEXP pattern [ ESCAPE escape-expression ]
```

Parameters

expression

The string to be searched.

pattern

The regular expression to search for within `expression`.

escape-expression

The escape character to be used in the match. The default is the backslash character (\).

Remarks

The REGEXP search condition matches a whole string, not a substring. To match on a substring with the string, enclose the string in wildcards that match the rest of the string (`.*pattern.*`). For example, `SELECT ... WHERE Description REGEXP 'car'` matches only car, not sportscar. However, `SELECT ... WHERE Description REGEXP '.*car'` matches car, sportscar, and any string that ends with car. Alternatively, you can rewrite your query to make use the REGEXP_SUBSTR function, which is designed to search for substrings within a string.

When matching against only a sub-character class, you must include the outer square brackets and the square brackets for the sub-character class. For example, `expression REGEXP '[[:digit:]]'`.

Database Collation and Matching

REGEXP only matches a literal in a pattern if it is the exact same character (that is, they have the same code point value). Ranges in character classes (for example, `'[A-F]'`) only match characters that code point values greater than or equal to the code point value of the first character in the range (A) and less than or equal to the code point value of the second character in the range (F).

Comparisons are performed character by character, unlike the equivalence (=) operator and other operators where the comparison is done string by string. For example, when a comparison is done in a UCA collation (CHAR or NCHAR with the collation set to UCA), `'Æ'='AE'` is true, but `'Æ' REGEXP 'AE'` is false.

National Character (NCHAR) Support

REGEXP search conditions can be used to compare CHAR and NCHAR strings. In this case, character set conversion is performed so that the comparison is done using a common data type. Then, a code point by code point comparison is performed.

You can specify `expression` or `pattern` as an NCHAR string literal by prefixing the quoted value with N (for example, `expression REGEXP N'pattern'`). You can also use the CAST function to cast the pattern to CHAR or NCHAR (for example, `expression REGEXP CAST(pattern AS datatype)`.

Standards

ANSI/ISO SQL Standard

The REGEXP search condition is not in the standard, but is roughly compatible with the LIKE_REGEX search condition of the ANSI/ISO SQL Standard, which is SQL language feature F841.

The software supports SQL Feature F281, which permits the pattern and escape-expressions to be arbitrary expressions evaluated at execution time. Feature F281 also permits `expression` to be an expression more complex than a simple column reference.

The software supports SQL Feature T042, which permits REGEXP search conditions to reference string-expressions that are LONG VARCHAR values.

REGEXP search conditions that specify NCHAR string expressions or patterns is feature F421.

Related Information

Comparisons Between CHAR and NCHAR [page 195]
String Literals [page 13]
Regular Expressions Overview [page 40]
CAST Function [Data Type Conversion] [page 275]
SIMILAR TO Search Condition [page 74]
LIKE Search Condition [page 67]
REGEXP_SUBSTR Function [String] [page 506]
Regular Expressions: Special Sub-character Classes [page 46]

1.1.7.6.3 SIMILAR TO Search Condition

Matches a pattern against a string.

Syntax

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE escape-expression ]
```

Parameters

expression

The expression to be searched.

pattern

The regular expression to search for within `expression`.

escape-expression

The escape character to use in the match. The default escape character is the null character, which can be specified in a string literal as '\x00'.

Regular expression syntax	Meaning
`\x`	Match anything that compares equal to `x`, where the escape character is assumed to be the backslash character (\). For example, `\[` matches '['.
`x`	Any character (other than a meta-character) matches itself. For example, `A` matches 'A'.

Remarks

To match a substring with the string, use the percentage sign wildcard (`%expression`). For example, `SELECT ... WHERE Description SIMILAR TO 'car'` matches only car, not sportscar. However, `SELECT ... WHERE Description SIMILAR TO '%car'` matches car, sportscar, and any string that ends with car.

When matching against only a sub-character class, you must include the outer square brackets, and the square brackets for the sub-character class. For example, `expression` SIMILAR TO '[[:digit:]]').

Comparisons are performed character by character, unlike the equivalence (=) operator and other operators where the comparison is done string by string. For example, when a comparison is done in a UCA collation (CHAR or NCHAR with the collation set to UCA), `'Æ'='AE'` is true, but `'Æ' SIMILAR TO 'AE'` is false.

For a character-by-character comparison to match, each single character in the expression being searched must match a single character or a wildcard in the SIMILAR TO pattern.

Database Collation and Matching

SIMILAR TO use the collation to determine character equivalence and evaluate character class ranges. For example, if the database is case- and accent-insensitive, matches are case- and accent-insensitive. Ranges are also evaluated using the collation sort order.

National Character (NCHAR) Support

SIMILAR TO search conditions can be used to compare CHAR and NCHAR strings. In this case, character set conversion is performed so that the comparison is done using a common data type. Then, a character-by-character comparison is performed.

You can specify `expression` or `pattern` as an NCHAR string literal by prefixing the quoted value with N (for example, `expression SIMILAR TO N'pattern'`). You can also use the CAST function to cast the pattern to CHAR or NCHAR (for example, `expression SIMILAR TO CAST(pattern AS datatype)`.

Standards

ANSI/ISO SQL Standard

The SIMILAR TO predicate is optional ANSI/ISO SQL Language Feature T141.

Related Information

Regular Expressions Overview [page 40]
Comparisons Between CHAR and NCHAR [page 195]
String Literals [page 13]
CAST Function [Data Type Conversion] [page 275]
REGEXP Search Condition [page 72]
LIKE Search Condition [page 67]
REGEXP_SUBSTR Function [String] [page 506]
Regular Expressions: Special Sub-character Classes [page 46]

1.1.7.7 IN Search Condition

Use the IN search condition to evaluate whether a value is found in a set.

Syntax

```
expression [ NOT ] IN { ( query-expression ) | ( expression-list ) }
```

Remarks

An IN search condition compares `expression` with the set of values returned by `query-expression` or the set of values specified in `expression-list`. Without the NOT keyword, the IN search condition evaluates according to the following rules:

- TRUE if `expression` is not NULL and equals at least one of the values.
- UNKNOWN if `expression` is NULL and the values list is not empty, or if at least one of the values is NULL and `expression` does not equal any of the other values.
- FALSE if `expression` is NULL and `query-expression` returns no values; or if `expression` is not NULL, none of the values are NULL, and `expression` does not equal any of the values.

The NOT keyword interchanges TRUE and FALSE.

The search condition `expression` IN (`expression-list`) is equivalent to `expression` = ANY (`expression-list`).

The search condition `expression` NOT IN (`expression-list`) is equivalent to `expression` <> ALL (`expression-list`).

The expressions in an `expression-list` can be a literal, variable, host variable, or a query expression whose result is a single row and a single column.

Standards

ANSI/ISO SQL Standard

Core Feature.

1.1.7.8 CONTAINS Search Condition

Use the CONTAINS search condition to evaluate whether a value is contained in a set.

Syntax

```
CONTAINS ( column-name [,...], contains-query-string )
```

```
contains-query-string :
simple-expression
| or-expression
```

```
simple-expression :
primary-expression
| and-expression
```

```
or-expression :
simple-expression { OR | | } contains-query-string
```

```
primary-expression :
basic-expression
| FUZZY" fuzzy-expression "
| and-not-expression

and-expression :
primary-expression [ AND | & ] simple-expression

and-not-expression :
primary-expression [ AND | & ] { NOT | - } basic-expression

basic-expression :
term
| phrase
| ( contains-query-string )
| near-expression
| before-expression

fuzzy-expression :
term
| fuzzy-expression term

term :
simple-term
| prefix-term

prefix-term :
simple-term*

phrase :
" phrase-string "

near-expression :
term NEAR [ [ min-distance ], max-distance ] term
| term { NEAR | ~ } term

before-expression :
term BEFORE [ [ min-distance ] max-distance ] term
| term BEFORE term

phrase-string :
term
| phrase-string term

simple-term : a string separated by whitespace and special characters that
represents a single indexed term (word) to search for

distance : a positive integer
```

Parameters

and-expression

Specifies that both `primary-expression` and `simple-expression` must be found in the text index.

By default, if no operator is specified between terms or expressions, an `and-expression` is assumed. For example, `'a b'` is interpreted as `'a AND b'`.

An ampersand (&) can be used instead of AND, and can abut the expressions or terms on either side (for example, `'a &b'`).

and-not-expression

Specifies that `primary-expression` must be present in the text index, but that `basic-expression` *must not* be found in the text index. This is also known as a **negation**.

If you use a hyphen for negation, the hyphen must have a space to the left of it, and must adjoin the term to the right; otherwise, the hyphen is not interpreted as a negation. For example, `'a -b'` is equivalent to `'a AND NOT b'`; whereas for `'a - b'`, the hyphen is ignored and the string is equivalent to `'a AND b'`. `'a-b'` is equivalent to the phrase `'"a b"'`.

or-expression

Specifies that at least one of `simple-expression` or `contains-query-string` must be present in the text index. For example, `'a|b'` is interpreted as `'a OR b'`.

fuzzy-expression

Finds terms that are similar to what you specify. Fuzzy matching is only supported on NGRAM text indexes.

near-expression

Searches for terms that are near each other. This is also known as a **proximity search**. For example, `'b NEAR[5] c'` searches for instances of b and c that are five or fewer terms away from each other. The order of terms is not significant; `'b NEAR c'` is equivalent to `'c NEAR b'`.

If the maximum distance is not specified, the default distance is 10. If the minimum distance is not specified, the default is 1.

The query `'apple NEAR[2, 10] tree'` matches the following documents:

```
'apple grows on the tree'
'apple and tree'
'tree and apple'
```

However, the query does not match the following documents:

```
'apple tree'
'tree apple'
```

You can specify a tilde (~) instead of NEAR. Using a tilde is equivalent to specifying NEAR without a distance, so a default of maximum 10 terms and minimum 1 term is applied. You cannot specify a maximum or minimum distance if you specify a tilde; it is always 10 terms.

NEAR expressions cannot be chained together (for example, `'a NEAR[1] b NEAR[1] c'` is invalid).

before-expression

Use `before-expression` to search for a term that is before another term. This is also known as a **proximity search**. For example, `'b BEFORE[5] c'` searches for instances of b that occur five or fewer terms before c. The order of terms is significant; `'b NEAR c'` is not equivalent to `'c NEAR b'`.

If the maximum distance is not specified, the default distance is 10. If the minimum distance is not specified, the default is 1.

The arguments in the matching text must be found in the same order as they are specified in the CONTAINS query string. For example, `'apple BEFORE[2, 10] tree'` matches the following documents:

```
'apple grows on the tree'
'apple and tree'
```

However, the query does not match the following documents:

```
'tree and apple'
'apple tree'
'tree apple'
```

The following queries are equivalent:

```
'apple BEFORE tree'
'apple BEFORE[10] tree'
'apple BEFORE[1, 10] tree'
```

BEFORE expressions cannot be chained together (for example, `'a BEFORE[1] b BEFORE[1] c'` is invalid).

prefix-term

Searches for terms that start with the specified prefix. For example, `'datab*'` searches for any term beginning with datab. This is also known as a **prefix search**. In a prefix search, matching is performed for the portion of the term to the left of the asterisk.

Remarks

The CONTAINS search condition takes a column list and `contains-query-string` as arguments. It can be used anywhere that a search condition (also referred to as predicate) can be specified, and returns TRUE or FALSE. The `contains-query-string` must be a constant string or a variable, with a value that is known at query time. The `contains-query-string` cannot be NULL, an empty string, or exceed 300 valid terms. A valid term is a term that is within the permitted term length and is not included in the STOPLIST. An error is returned when the `contains-query-string` exceeds 300 valid terms.

If the text configuration settings cause all of the terms in the `contains-query-string` to be dropped, the result of the CONTAINS search condition is FALSE.

If multiple columns are specified, then they must all resolve to a single base table (a text index cannot span multiple base tables). The base table can be referenced directly in the FROM clause, or it can be used in a view or derived table. If `column-name` is a column from a view or derived table, the CONTAINS search condition is recursively pushed into the nested query blocks of the query expression provided that the following is true:

- The columns of the query expressions referenced in the original CONTAINS search condition all resolve to a single base table in the nested query blocks.
- If more than one text index is used in the nested query blocks, they must have the same text index configuration.
- The nested query blocks cannot contain TOP, LIMIT, FIRST, or window aggregates.
- The original CONTAINS must be in the WHERE clause in a conjunctive predicate. The columns referenced in the original CONTAINS search condition must resolve to columns of the same view or derived table.

The following warnings apply to the use of non-alphanumeric characters in query strings:

- An asterisk in the middle of a term returns an error.
- Do not use non-alphanumerics (including special characters) in `fuzzy-expression` because they are treated as whitespace and serve as term breakers.
- If possible, do not include non-alphanumeric characters that are not special characters in your query string. Any non-alphanumeric character that is not a special character causes the term containing it to be treated as a phrase, breaking the term at the location of the character. For example, `'things we've done'` is interpreted as `'things "we ve" done'`.

Within phrases, the asterisk is the only special character that continues to be interpreted as a special character. All other special characters within phrases are treated as whitespace and serve as term breakers.

Interpretation of `contains-query-string` takes place in two main steps:

Step 1: Interpreting operators and precedence

During this step, keywords are interpreted as operators, and rules of precedence are applied.

Step 2: Applying text configuration object settings

During this step, the text configuration object settings are applied to terms. For example, on an NGRAM text index, terms are broken down into their n-gram representation. During this step, the query terms that exceed the term length settings, or that are in the stoplist, are dropped.

Operator Precedence in a CONTAINS Search Condition

During query evaluation, expressions are evaluated using the following order of precedence:

1. BEFORE, FUZZY, NEAR
2. AND NOT
3. AND
4. OR

Allowed Syntax for Asterisk (*)

The asterisk is used for **prefix searching**. An asterisk can occur at the end of the query string, or be followed by a space, ampersand, vertical bar, closing bracket, or closing quotation mark. Any other usage of asterisk returns an error.

The following table shows allowable asterisk usage:

Query string	Equivalent to	Interpreted as
`'th*'`		Find any term beginning with th.
`'th*&best'`	`'th* AND best'` and `'th* best'`	Find any term beginning with th, and the term best.

Query string	Equivalent to	Interpreted as
`'th*|best'`	`'th* OR best'`	Find either any term beginning with th, or the term best.
`'very&(best|th*)'`	`'very AND (best OR th*)'`	Find the term very, and the term best or any term beginning with th.
`'"fast auto*"'`		Find the term fast, immediately followed by a term beginning with auto.
`'"auto* price"'`		Find a term beginning with auto, immediately followed by the term price.

> **Note**
>
> Interpretation of query strings containing asterisks can vary depending on the text configuration object settings.

Allowed Syntax for Hyphen (-)

The hyphen can be used for term or expression **negation** and is equivalent to NOT. Whether a hyphen is interpreted as a negation depends on its location in the query string. For example, when a hyphen immediately precedes a term or expression, it is interpreted as a negation. If the hyphen is embedded within a term, it is interpreted as a hyphen.

A hyphen used for negation must be preceded by a whitespace and followed immediately by an expression.

When used in a phrase of a fuzzy expression, the hyphen is treated as whitespace and used as a term breaker.

The following table shows the allowed syntax for hyphen:

Query String	Equivalent to:	Interpreted as:
`'the -best'`	`'the AND NOT best'`, `'the AND -best'`, `'the & -best'`, `'the NOT best'`	Find the term the, and not the term best.
`'the -(very best)'`	`'the AND NOT (very AND best)'`	Find the term the, and not the terms very and best.
`'the -"very best"'`	`'the AND NOT "very best"'`	Find the term the, and not the phrase very best.
`'alpha-numerics'`	`'"alpha numerics"'`	Find the term alpha, immediately followed by the term numerics.
`'wild - west'`	`'wild west'`, and `'wild AND west'`	Find the term wild, and the term west.

Allowed Syntax for Special Characters

The following table shows the allowed syntax for all special characters except asterisk and hyphen.

These characters are not considered special characters if they are found in a phrase, and are dropped.

> **i** Note
>
> The same restrictions with regards to specifying string literals also apply to the query string. For example, apostrophes must be escaped, and so on.

Character or Syntax	Usage Examples and Remarks
ampersand (&)	The ampersand is equivalent to AND and can be specified as follows: • `'a & b'` • `'a &b'` • `'a&b'` • `'a& b'`
vertical bar (|)	The vertical bar is equivalent to OR and can be specified as follows: • `'a|b'` • `'a |b'` • `'a | b'` • `'a| b'`
double quotes (")	Double quotes are used to contain a sequence of terms where order and relative distance are important. For example, in the query string **`'learn "full text search"'`**, **`full text search`** is a phrase. In this example, learn can come before or after the phrase, or exist in another column (if the text index is built on more than one column), but the exact phrase must be found in a single column.
parentheses ()	Parentheses are used to specify the order of evaluation of expressions if different from the default order. For example **`'a AND (b|c)'`** is interpreted as a, and b or c.
tilde (~)	The tilde is equivalent to NEAR[10]. The query string **`'full~text'`** is equivalent to **`'full NEAR text'`**, and is interpreted as: the term full within ten terms of the term text. You cannot specify a distance with the tilde.
square brackets [ ]	Square brackets are used with the keywords BEFORE or NEAR to contain `distance`. Other uses of square brackets return an error.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Example Text Configuration Objects
Full Text Proximity Searches
What to Specify When Creating or Altering Text Configuration Objects
Full Text Searches on Views
Full Text Prefix Searches
String Literals [page 13]
Full Text Search
What to Specify When Creating or Altering Text Configuration Objects
FROM Clause [page 1173]
sa_char_terms System Procedure [page 1534]
sa_nchar_terms System Procedure [page 1664]

1.1.7.9 EXISTS Search Condition

Use the EXISTS search condition to evaluate whether a value is found in a set.

Syntax

```
EXISTS ( subquery )
```

Remarks

The EXISTS search condition is TRUE if the subquery result contains at least one row, and FALSE if the subquery result does not contain any rows. The EXISTS search condition cannot be UNKNOWN.

Standards

ANSI/ISO SQL Standard

Core Feature.

1.1.7.10 IS NULL and IS NOT NULL Search Conditions

Use the IS NULL search condition to evaluate whether a value in a set is NULL.

Syntax

```
expression IS [ NOT ] NULL
```

Remarks

Without the NOT keyword, the IS NULL search condition is TRUE if the expression is the NULL value, and FALSE otherwise. The NOT keyword reverses the meaning of the search condition.

Standards

ANSI/ISO SQL Standard

Core Feature.

1.1.7.11 Truth Value Search Conditions

Use the IS TRUE search condition to evaluate whether a condition evaluates to a specified value.

Syntax

```
IS [ NOT ] truth-value
```

Remarks

Without the NOT keyword, the search condition is TRUE if the `condition` evaluates to the supplied `truth-value`, which must be one of TRUE, FALSE, or UNKNOWN. Otherwise, the value is FALSE. The NOT keyword reverses the meaning of the search condition, but leaves UNKNOWN unchanged.

Standards

ANSI/ISO SQL Standard

Truth value search conditions comprise optional ANSI/ISO SQL Language Feature F571.

1.1.7.12 Three-valued Logic

The following tables display how the AND, OR, NOT, and IS logical operators of SQL work in three-valued logic.

AND Operator

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR Operator

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT Operator

TRUE	FALSE	UNKNOWN
FALSE	TRUE	UNKNOWN

IS Operator

IS	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE
UNKNOWN	FALSE	FALSE	TRUE

Standards

ANSI/ISO SQL Standard

Core Feature. Truth value tests, such as IS UNKNOWN, comprise SQL Language Feature F571.

Related Information

NULL Special Value [page 103]

1.1.7.13 Explicit Selectivity Estimates

The database server uses statistical information to determine the most efficient strategy for executing each statement.

The database server automatically gathers and updates these statistics. These statistics are stored permanently in the database in the system table ISYSCOLSTAT. Statistics gathered while processing one statement are available when searching for efficient ways to execute subsequent statements.

Occasionally, the statistics may become inaccurate or relevant statistics may be unavailable. This condition is most likely to arise when few queries have been executed since a large amount of data was added, updated, or deleted. In this situation, you may want to execute a CREATE STATISTICS statement.

If there are problems with a particular execution plan, you can use optimizer hints to require that a particular index be used.

In unusual circumstances, however, these measures may prove ineffective. In such cases, you can sometimes improve performance by supplying explicit selectivity estimates.

For each table in a potential execution plan, the optimizer must estimate the number of rows that will be part of the result set. If you know that a condition has a success rate that differs from the optimizer's estimate, you can explicitly supply a user estimate in the search condition.

The estimate is a percentage. It can be a positive integer or can contain fractional values.

> ⚠ Caution
>
> Whenever possible, avoid supplying explicit estimates in statements that are to be used on an ongoing basis. Should the data change, the explicit estimate may become inaccurate and may force the optimizer to select poor plans. If you do use explicit selectivity estimates, ensure that the number is accurate. Do not, for example, supply values of 0% or 100% to force the use of an index.

You can disable user estimates by setting the database option user_estimates to Off. The default value for user_estimates is Override-Magic, which means that user-supplied selectivity estimates are used only when the optimizer would use a MAGIC (default) selectivity value for the condition. The optimizer uses MAGIC values as a last resort when it is unable to accurately predict the selectivity of a predicate.

Example

The following query provides an estimate that one percent of the ShipDate values are later than 2001/06/30:

```
SELECT  ShipDate
   FROM  GROUPO.SalesOrderItems
WHERE ( ShipDate > '2001/06/30', 1 )
ORDER BY ShipDate DESC;
```

The following query estimates that half a percent of the rows satisfy the condition:

```
SELECT *
   FROM GROUPO.Customers c, GROUPO.SalesOrders o
WHERE (c.ID = o.CustomerID, 0.5);
```

Fractional values enable more accurate user estimates for joins and large tables.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Optimizer Estimates and Statistics
CREATE STATISTICS Statement [page 991]
FROM Clause [page 1173]
user_estimates Option

1.1.8 Special Values

Special values can be used in expressions, and as column defaults when creating tables.

While some special values can be queried, some can only be used as default values for columns. For example, LAST USER, TIMESTAMP and UTC TIMESTAMP can only be used as default values.

In this section:

CURRENT PUBLISHER returns a string that contains the publisher user ID of the database for SQL Remote replications.

CURRENT REMOTE USER Special Value [page 92]
If the current connection belongs to the receive phase of SQL Remote, then CURRENT REMOTE USER returns the user ID of the remote user that created the messages that are currently being applied on this connection. In all other circumstances, CURRENT REMOTE USER is a NULL value.

CURRENT SERVER DATE Special Value [page 93]
CURRENT DATE returns the present year, month, and day in the time zone of the database server.

CURRENT SERVER TIME Special Value [page 93]
CURRENT SERVER TIME returns the present hour, minute, second, and fraction of a second in the time zone of the database server.

CURRENT SERVER TIMESTAMP Special Value [page 94]
CURRENT SERVER TIMESTAMP combines CURRENT SERVER DATE and CURRENT SERVER TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second, and fraction of a second in the time zone of the database server.

CURRENT TIME Special Value [page 95]
CURRENT TIME returns the present hour, minute, second, and fraction of a second.

CURRENT TIMESTAMP Special Value [page 96]
CURRENT TIMESTAMP combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second and fraction of a second.

CURRENT USER Special Value [page 98]
CURRENT USER contains the user ID of the current connection.

CURRENT UTC TIMESTAMP Special Value [page 99]
CURRENT UTC TIMESTAMP returns the Coordinated Universal Time (UTC) containing the year, month, day, hour, minute, second, fraction of a second, and time zone.

EXECUTING USER Special Value [page 100]
SQL special value that returns the current effective user.

INVOKING USER Special Value [page 101]
SQL special value that returns the user that invoked the current procedure, or returns the current logged in user if no procedure is executing.

LAST USER Special Value [page 102]
LAST USER is the user ID of the user who last modified the row.

NULL Special Value [page 103]
The NULL value specifies a value that is unknown or not applicable.

PROCEDURE OWNER Special Value [page 106]
SQL special value that returns the owner of the current procedure, or NULL if queried outside of a procedure context.

SESSION USER Special Value [page 107]
SQL special value that stores the user that is currently logged in.

SQLCODE Special Value [page 108]
SQLCODE indicates the disposition of the most recently executed SQL statement.

SQLSTATE Special Value [page 109]

SQLSTATE indicates whether the most recently executed SQL statement resulted in a success, error, or warning condition.

TIMESTAMP Special Value [page 111]
The TIMESTAMP default value is used to record the local date and time of day when a row in a table was last modified.

USER Special Value [page 113]
USER contains the user ID of the current connection.

UTC TIMESTAMP Special Value [page 113]
The UTC TIMESTAMP default value is used to record the Coordinated Universal Time (UTC) when a row in a table was last modified.

1.1.8.1 CURRENT DATABASE Special Value

CURRENT DATABASE returns the name of the current database.

Data type

string

Remarks

During an UPDATE operation, columns with a default value of CURRENT DATABASE are not changed.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Expressions in SQL Statements [page 34]

1.1.8.2 CURRENT DATE Special Value

CURRENT DATE returns the present year, month, and day.

Data type

DATE

Remarks

When the database is using a simulated time zone, the simulated time zone is used to calculate this value.

During an UPDATE operation, columns with a default value of CURRENT DATE are not changed.

Standards

ANSI/ISO SQL Standard

Not in the ANSI/ISO SQL Standard. In the standard, the special register that defines the present date is called CURRENT_DATE. The software does not support CURRENT_DATE.

Example

Create the Australian Eastern Time zone.

```
CREATE TIME ZONE NewSouthWales OFFSET '10:00'
STARTING 'Oct/Sun>=1' AT '2:00'
ENDING 'Apr/Sun>=1' AT '2:00';
```

Create a table with a DEFAULT column set to the CURRENT DATE special value. Then insert some values under two different time zones.

```
CREATE OR REPLACE TABLE TEST( COL1 DATE DEFAULT CURRENT DATE,COL2 INT);
SET OPTION PUBLIC.time_zone='NewSouthWales';
INSERT INTO TEST(COL2) VALUES(1),(2);
SET OPTION PUBLIC.time_zone=;
INSERT INTO TEST(COL2) VALUES(3),(4);
SELECT * FROM TEST;
```

The first two rows are inserted using the current date in the NewSouthWales time zone and the last two rows are inserted using the current date in the server's time zone.

Related Information

DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]

1.1.8.3 CURRENT PUBLISHER Special Value

CURRENT PUBLISHER returns a string that contains the publisher user ID of the database for SQL Remote replications.

Data Type

string

Remarks

The publisher is set using the PUBLIC.db_publisher option, or by using the GRANT PUBLISH and REVOKE PUBLISH statements.

CURRENT PUBLISHER can be used as a default value in columns with character data types.

When an update or insert operation is performed on a column defined as DEFAULT CURRENT PUBLISHER, the column is updated with the current value of CURRENT PUBLISHER.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Viewing the Publisher (SQL Central)
db_publisher Option
GRANT PUBLISH Statement [SQL Remote] [page 1206]
REVOKE PUBLISH Statement [SQL Remote] [page 1349]

1.1.8.4 CURRENT REMOTE USER Special Value

If the current connection belongs to the receive phase of SQL Remote, then CURRENT REMOTE USER returns the user ID of the remote user that created the messages that are currently being applied on this connection. In all other circumstances, CURRENT REMOTE USER is a NULL value.

Data Type

string

Remarks

The CURRENT REMOTE USER special value is set by the receive phase of SQL Remote when it is applying messages to the database. The CURRENT REMOTE USER special value is most useful in triggers to determine whether the operations being applied are being applied by the receive phase of SQL Remote, and if they are, which remote user generated the operations being applied.

When an update or insert operation is performed on a column defined as DEFAULT CURRENT REMOTE USER, the column is updated with the current value of CURRENT REMOTE USER.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

The CURRENT REMOTE USER Special Value
SQL Remote Message Agent Utility (dbremote)

1.1.8.5 CURRENT SERVER DATE Special Value

CURRENT DATE returns the present year, month, and day in the time zone of the database server.

Data Type

DATE

Remarks

During an UPDATE operation, the database server does not change columns with a default value of CURRENT SERVER DATE.

Standards

ANSI/ISO SQL Standard

Not in the ANSI/ISO SQL Standard. In the standard, the special register that defines the present date is called CURRENT_SERVER_DATE. The software does not support CURRENT_SERVER_DATE.

1.1.8.6 CURRENT SERVER TIME Special Value

CURRENT SERVER TIME returns the present hour, minute, second, and fraction of a second in the time zone of the database server.

Data Type

TIME

Remarks

The fraction of a second is stored to six decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

During an UPDATE operation, the database server does not change columns with a default value of CURRENT SERVER TIME.

The CURRENT SERVER TIME is not affected by a simulated time zone using the time_zone option setting.

Standards

ANSI/ISO SQL Standard

Not in the ANSI/ISO SQL Standard. In the standard, the special register that defines the present time is called CURRENT_SERVER_TIME. The software does not support CURRENT_SERVER_TIME.

1.1.8.7 CURRENT SERVER TIMESTAMP Special Value

CURRENT SERVER TIMESTAMP combines CURRENT SERVER DATE and CURRENT SERVER TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second, and fraction of a second in the time zone of the database server.

Data Type

TIMESTAMP

Remarks

The fraction of a second is stored to six decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

Standards

ANSI/ISO SQL Standard

Not in the ANSI/ISO SQL Standard. In the standard, the special register that defines the present timestamp is called CURRENT_TIMESTAMP.

1.1.8.8 CURRENT TIME Special Value

CURRENT TIME returns the present hour, minute, second, and fraction of a second.

Data Type

TIME

Remarks

The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

When the database is using a simulated time zone, the simulated time zone is used to calculate this value.

During an UPDATE operation, columns with a default value of CURRENT TIME are not changed.

Standards

ANSI/ISO SQL Standard

Not in the ANSI/ISO SQL Standard. In the standard, the special register that defines the present time is called CURRENT_TIME. The software does not support CURRENT_TIME.

Example

Create the Australian Eastern Time zone.

```
CREATE TIME ZONE NewSouthWales OFFSET '10:00'
STARTING 'Oct/Sun>=1' AT '2:00'
ENDING 'Apr/Sun>=1' AT '2:00';
```

Create a table with a DEFAULT column set to the CURRENT TIME special value. Then insert some values under two different time zones.

```
CREATE OR REPLACE TABLE TEST( COL1 TIME DEFAULT CURRENT TIME,COL2 INT);
SET OPTION PUBLIC.time_zone='NewSouthWales';
INSERT INTO TEST(COL2) VALUES(1),(2);
SET OPTION PUBLIC.time_zone=;
INSERT INTO TEST(COL2) VALUES(3),(4);
SELECT * FROM TEST;
```

The first two rows are inserted using the current time in the NewSouthWales time zone and the last two rows are inserted using the current time in the server's time zone.

Related Information

Expressions in SQL Statements [page 34]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP Special Value [page 111]
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.1.8.9 CURRENT TIMESTAMP Special Value

CURRENT TIMESTAMP combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second and fraction of a second.

Data Type

TIMESTAMP

Remarks

The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

Unlike DEFAULT TIMESTAMP, columns declared with DEFAULT CURRENT TIMESTAMP do not necessarily contain unique values. If uniqueness is required, consider using DEFAULT TIMESTAMP instead.

When the database is using a simulated time zone, the simulated time zone is used to calculate this value.

The information CURRENT TIMESTAMP returns is equivalent to the information returned by the GETDATE and NOW functions.

CURRENT_TIMESTAMP is equivalent to CURRENT TIMESTAMP.

> **i** Note
>
> The main difference between DEFAULT CURRENT TIMESTAMP and DEFAULT TIMESTAMP is that DEFAULT CURRENT TIMESTAMP columns are set only at INSERT (unless explicitly set at UPDATE), while DEFAULT TIMESTAMP columns are set at both INSERT and UPDATE.

Standards

ANSI/ISO SQL Standard

Not in the ANSI/ISO SQL Standard. In the standard, the special register that defines the present timestamp is called CURRENT_TIMESTAMP.

Example

Create the Australian Eastern Time zone.

```
CREATE TIME ZONE NewSouthWales OFFSET '10:00'
STARTING 'Oct/Sun>=1' AT '2:00'
ENDING 'Apr/Sun>=1' AT '2:00';
```

Create a table with a DEFAULT column set to the CURRENT TIMESTAMP special value. Then insert some values under two different time zones.

```
CREATE OR REPLACE TABLE TEST( COL1 TIMESTAMP DEFAULT CURRENT TIMESTAMP,COL2 INT);
SET OPTION PUBLIC.time_zone='NewSouthWales';
INSERT INTO TEST(COL2) VALUES(1),(2);
SET OPTION PUBLIC.time_zone=;
INSERT INTO TEST(COL2) VALUES(3),(4);
SELECT * FROM TEST;
```

The first two rows are inserted using the current date and time in the NewSouthWales time zone and the last two rows are inserted using the current date and time in the server's time zone.

Related Information

Expressions in SQL Statements [page 34]
CURRENT TIME Special Value [page 95]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]

DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP Special Value [page 111]
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.1.8.10 CURRENT USER Special Value

CURRENT USER contains the user ID of the current connection.

Data Type

string

Remarks

CURRENT USER can be used as a default value in columns with character data types.

On UPDATE, columns with the CURRENT USER default are not changed unless explicitly updated. The LAST USER default can be used to track updates by users.

CURRENT_USER is equivalent to CURRENT USER.

Standards

ANSI/ISO SQL Standard

Not in the ANSI/ISO SQL Standard. In the standard, the special register that defines the current user is called CURRENT_USER.

Related Information

Expressions in SQL Statements [page 34]
LAST USER Special Value [page 102]
USER Special Value [page 113]

1.1.8.11 CURRENT UTC TIMESTAMP Special Value

CURRENT UTC TIMESTAMP returns the Coordinated Universal Time (UTC) containing the year, month, day, hour, minute, second, fraction of a second, and time zone.

Data Type

TIMESTAMP WITH TIME ZONE

Remarks

This feature allows data to be entered with a consistent time reference, regardless of the time zone in which the data was entered.

During an UPDATE operation, columns with the CURRENT UTC TIMESTAMP default value are not changed unless explicitly updated.

The UTC TIMESTAMP default can be used to track the UTC time of updates.

Standards

ANSI/ISO SQL Standard

Not in the standard. However, the TIMESTAMP WITH TIME ZONE data type is optional ANSI/ISO SQL Language Feature F41.

Related Information

Expressions in SQL Statements [page 34]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
DATE Data Type [page 168]

DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP Special Value [page 111]
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.1.8.12 EXECUTING USER Special Value

SQL special value that returns the current effective user.

Data Type

STRING

Remarks

Use EXECUTING USER, INVOKING USER, SESSION USER, and PROCEDURE OWNER to determine which users can execute, and are executing, procedures and user-defined functions. Depending on how many layers of nesting a particular procedure call has, and based on whether the previous and current procedure are SQL SECURITY DEFINER or SQL SECURITY INVOKER, the EXECUTING USER, and INVOKING USER can and do change.

EXECUTING_USER is equivalent to EXECUTING USER.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Procedures and Functions Running with Owner or Invoker Privileges
INVOKING USER Special Value [page 101]
PROCEDURE OWNER Special Value [page 106]
SESSION USER Special Value [page 107]

1.1.8.13 INVOKING USER Special Value

SQL special value that returns the user that invoked the current procedure, or returns the current logged in user if no procedure is executing.

Data Type

STRING

Remarks

Use INVOKING USER, SESSION USER, EXECUTING USER, and PROCEDURE OWNER to determine which users can execute, and are executing, procedures and user-defined functions. Depending on how many layers of nesting a particular procedure call has, and based on whether the previous and current procedure are SQL SECURITY DEFINER or SQL SECURITY INVOKER, the INVOKING USER and EXECUTING USER can and do change.

INVOKING_USER is equivalent to INVOKING USER.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Procedures and Functions Running with Owner or Invoker Privileges
EXECUTING USER Special Value [page 100]
PROCEDURE OWNER Special Value [page 106]

SESSION USER Special Value [page 107]

1.1.8.14 LAST USER Special Value

LAST USER is the user ID of the user who last modified the row.

Data Type

String

Remarks

LAST USER can be used as a default value in columns with character data types.

On INSERT, this constant has the same effect as CURRENT USER.

On UPDATE, if a column with a default value of LAST USER is not explicitly modified, it is changed to the name of the current user.

When combined with the DEFAULT TIMESTAMP, a default value of LAST USER can be used to record (in separate columns) both the user and the date and time of day a row was last changed.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

CREATE TABLE Statement [page 1002]
CURRENT USER Special Value [page 98]
CURRENT TIMESTAMP Special Value [page 96]
USER Special Value [page 113]

1.1.8.15 NULL Special Value

The NULL value specifies a value that is unknown or not applicable.

Syntax

NULL

Remarks

NULL is a special value that is different from any valid value for any data type. However, the NULL value is a legal value in any data type. NULL is used to represent missing or inapplicable information. There are two separate and distinct cases where NULL is used:

Situation	Description
missing	The field does have a value, but that value is unknown.
inapplicable	The field does not apply for this particular row.

SQL allows columns to be created with the NOT NULL restriction. Those particular columns cannot contain NULL.

The NULL value introduces the concept of three valued logic to SQL. The NULL value compared using any comparison operator with any value (including the NULL value) is "UNKNOWN." The only search condition that returns TRUE is the IS NULL predicate. In SQL, rows are selected only if the search condition in the WHERE clause evaluates to TRUE; rows that evaluate to UNKNOWN or FALSE are not selected.

Column space utilization for NULL values is 1 bit per column and space is allocated in multiples of 8 bits. The NULL bit usage is fixed based on the number of columns in the table that allow NULL values.

The IS [NOT] `truth-value` clause, where `truth-value` is one of TRUE, FALSE or UNKNOWN can be used to select rows where the NULL value is involved.

In the following examples, the column Salary contains NULL.

Condition	Truth Value	Selected?
Salary = NULL	UNKNOWN	NO
Salary <> NULL	UNKNOWN	NO
NOT (Salary = NULL)	UNKNOWN	NO
NOT (Salary <> NULL)	UNKNOWN	NO
Salary = 1000	UNKNOWN	NO
Salary IS NULL	TRUE	YES
Salary IS NOT NULL	FALSE	NO
Salary = `expression` IS UNKNOWN	TRUE	YES

The same rules apply when comparing columns from two different tables. Therefore, joining two tables together does not select rows where any of the columns compared contain the NULL value.

NULL also has an interesting property when used in numeric expressions. The result of any numeric expression involving the NULL value is NULL. If NULL is added to a number, then the result is NULL, not a number. To treat NULL as 0, use the ISNULL(`expression`, 0) function.

Many common errors in formulating SQL queries are caused by the behavior of NULL. You have to be careful to avoid these problem areas.

Set Operators and DISTINCT Clause

In SQL, comparisons to NULL within search conditions yield UNKNOWN as the result. However, when determining whether two rows are duplicates of each other, SQL treats NULL as equivalent to NULL. These semantics apply to the set operators (UNION, INTERSECT, EXCEPT), GROUP BY, PARTITION within a WINDOW clause, and SELECT DISTINCT.

For example, if a column called redundant contained NULL for every row in a table T1, then the following statement would return a single row:

```
SELECT DISTINCT redundant FROM T1;
```

You can also use the IS DISTINCT FROM and IS NOT DISTINCT FROM sargable search conditions to determine whether two expressions are equal or if both expressions are NULL.

Prerequisites

Must be connected to the database.

Side Effects

None.

Standards

ANSI/ISO SQL Language

Core Feature.

Transact-SQL

In some contexts, Adaptive Server Enterprise treats comparisons to NULL values differently. If an `expression` is compared to a variable or NULL literal using equality or inequality, and if `expression` is a simple expression that refers to the column of a base table or view, then the comparison is performed

using two-valued logic, with NULL = NULL yielding TRUE rather than UNKNOWN. The list of possible comparisons with these semantics, and their ANSI/ISO SQL Standard equivalents, are as follows:

Transact-SQL Comparison	ANSI/ISO SQL Standard Equivalent
`expression` = NULL	`expression` IS NULL
`expression` != NULL	NOT (`expression` IS NULL)
`expression` = `variable`	`expression` = `variable` IS TRUE OR (`expression` IS NULL AND `variable` IS NULL)
`expression` != `variable`	`expression` != `variable` IS TRUE AND (NOT `expression` IS NULL OR NOT `variable` IS NULL)

SQL Anywhere will implement these semantics to match Adaptive Server Enterprise behavior if the ansinull option is set to OFF. The ansinull option is set to OFF by default for Open Client and jConnect connections. To ensure ANSI/ISO SQL Standard semantics, you can either reset the ansinull option to ON, or use an IS [NOT] NULL predicate instead of an equality comparison.

Unique indexes in SQL Anywhere can hold rows that hold NULL and are otherwise identical. Adaptive Server Enterprise does not permit such entries in unique indexes.

If you use jConnect, the tds_empty_string_is_null option controls whether empty strings are returned as NULL strings or as a string containing one blank character.

Example

The following INSERT statement inserts a NULL into the date_returned column of the Borrowed_book table.

```
INSERT INTO Borrowed_book ( date_borrowed, date_returned, book )
VALUES ( CURRENT DATE, NULL, '1234' );
```

Related Information

tds_empty_string_is_null Option
Expressions in SQL Statements [page 34]
ansinull Option
tds_empty_string_is_null Option
Search Conditions [page 58]
IS DISTINCT FROM and IS NOT DISTINCT FROM Search Conditions [page 63]

1.1.8.16 PROCEDURE OWNER Special Value

SQL special value that returns the owner of the current procedure, or NULL if queried outside of a procedure context.

Data Type

STRING

Remarks

Use PROCEDURE OWNER, INVOKING USER, SESSION USER, and EXECUTING USER to determine which users can execute, and are executing, procedures and user-defined functions. Depending on how many layers of nesting a particular procedure call has, and based on whether the previous and current procedure are SQL SECURITY DEFINER or SQL SECURITY INVOKER, the EXECUTING USER and INVOKING USER can and do change.

PROCEDURE_OWNER is equivalent to PROCEDURE OWNER.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Procedures and Functions Running with Owner or Invoker Privileges
EXECUTING USER Special Value [page 100]
INVOKING USER Special Value [page 101]
SESSION USER Special Value [page 107]

1.1.8.17 SESSION USER Special Value

SQL special value that stores the user that is currently logged in.

Data Type

STRING

Remarks

Use SESSION USER, INVOKING USER, EXECUTING USER, and PROCEDURE OWNER to determine which users can execute, and are executing, procedures and user-defined functions. Depending on how many layers of nesting a particular procedure call has, and based on whether the previous and current procedure are SQL SECURITY DEFINER or SQL SECURITY INVOKER, the INVOKING USER, and EXECUTING USER can and do change. However, SESSION USER always remains the logged in user.

SESSION_USER is equivalent to SESSION USER.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Procedures and Functions Running with Owner or Invoker Privileges
EXECUTING USER Special Value [page 100]
INVOKING USER Special Value [page 101]
PROCEDURE OWNER Special Value [page 106]

1.1.8.18 SQLCODE Special Value

SQLCODE indicates the disposition of the most recently executed SQL statement.

Data Type

Signed INTEGER

Remarks

The database server sets a SQLSTATE and SQLCODE for each SQL statement it executes. SQLCODEs are product-specific (for example, MobiLink has its own SQLCODEs), and can be used to learn additional information about the SQLSTATE. For example, positive values other than 100 indicate product-specific `warning` conditions. Negative values indicate product-specific `exception` conditions. The value 100 indicates "no data" (for example, at the end of a result set fetched via a cursor).

SQLSTATE and SQLCODE are related in that each SQLCODE corresponds to a SQLSTATE, and each SQLSTATE can correspond to one or more SQLCODEs.

To return the error condition associated with a SQLCODE, you can use the ERRORMSG function.

> **i** Note
>
> SQLSTATE is the preferred status indicator for the outcome of a SQL statement.

Standards

ANSI/ISO SQL Standard

Not in the standard. SQLSTATE is the preferred status indicator.

Related Information

SQLSTATE Special Value [page 109]
SQL Anywhere Error Messages Sorted by SQLCODE
Expressions in SQL Statements [page 34]
ERRORMSG Function [Miscellaneous] [page 364]

1.1.8.19 SQLSTATE Special Value

SQLSTATE indicates whether the most recently executed SQL statement resulted in a success, error, or warning condition.

Data Type

String

Remarks

The database server sets a SQLSTATE and SQLCODE for each SQL statement it executes. A SQLSTATE is a string that indicates the whether the most recently executed SQL statement resulted in a success, warning, or error condition.

Each SQLSTATE represents errors that are common to all platforms, and usually contain non-product-specific wording. The format of a SQLSTATE value is a two-character class value, followed by a three-character subclass value. Guidelines for SQLSTATE conformance with regard to class and subclass values are outlined in the ISO/ANSI SQL standard.

The database server conforms to the ISO/ANSI SQLSTATE conventions with the following additions and exceptions:

Class and Subclass	Condition
01WC*x*	Warnings related to character set conversion
38*xxx*	External function exception
42X*xx*	Syntax error: expressions
42R*xx*	Syntax error: referential integrity (for example, attempt to create second primary key)
42W*xx*	Syntax error: generic
42U*xx*	Syntax error: duplicate, undefined, or ambiguous object reference
42Z*xx*	Access violation
54W*xx*	Product limit exceeded
55W*xx*	Object not in required state for operation to succeed
57*xxx*	Resource not available or operator intervention
5R*xxx*	SQL Remote errors
WB*xxx*	Online backup errors
WI*xxx*	Internal database errors

Class and Subclass	Condition
WP`xxx`	Errors in procedures, variables, and so on
WL`xxx`	Errors loading and/or unloading
WW`xxx`	Miscellaneous SQL Anywhere-specific errors/warnings (including system failures)
WO`xxx`	Remote data access feature-related errors
WJ`xxx`	JCS and JDBC related errors
WC`xxx`	Character translation errors
WX`xxx`	XML-related errors
WT`xxx`	Text-related errors

The successful completion class is '00`xxx`' (for example, '00000').

SQLSTATE and SQLCODE are related in that each SQLCODE corresponds to a SQLSTATE, and each SQLSTATE can correspond to one or more SQLCODEs.

To return the error condition associated with a SQLSTATE, you can use the ERRORMSG function.

Standards

ANSI/ISO SQL Standard

SQLSTATE classes (the first two characters) beginning with the values '0'-'4', and 'A'-'H' are defined by the ANSI/ISO SQL Standard. Other classes are implementation-defined. Similarly, subclass values that begin with values '0'-'4', and 'A'-'H' are defined by the ANSI/ISO SQL Standard. Subclass values outside these ranges are implementation-defined.

Related Information

ERRORMSG Function [Miscellaneous] [page 364]
SQLCODE Special Value [page 108]
SQL Anywhere Error Messages Sorted by SQLSTATE
Expressions in SQL Statements [page 34]

1.1.8.20 TIMESTAMP Special Value

The TIMESTAMP default value is used to record the local date and time of day when a row in a table was last modified.

Data Type

TIMESTAMP

Remarks

The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the present date and time of day whenever the row is updated.

When the database is using a simulated time zone, the simulated time zone is used to calculate this value.

Columns declared with DEFAULT TIMESTAMP contain unique values so that applications can detect near-simultaneous updates to the same row. If the present timestamp value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate timestamp values with the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable @@dbts returns a TIMESTAMP value corresponding to the last value generated for any column using DEFAULT TIMESTAMP in the database. When the database is using a simulated time zone, the TIMESTAMP value is relative to that time zone.

> **i** Note
>
> The main difference between DEFAULT TIMESTAMP and DEFAULT CURRENT TIMESTAMP is that DEFAULT CURRENT TIMESTAMP columns are set only at INSERT (unless explicitly set at UPDATE), while DEFAULT TIMESTAMP columns are set at both INSERT and UPDATE.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Create the Australian Eastern Time zone.

```
CREATE TIME ZONE NewSouthWales OFFSET '10:00'
STARTING 'Oct/Sun>=1' AT '2:00'
ENDING 'Apr/Sun>=1' AT '2:00';
```

Create a table with a DEFAULT column set to the CURRENT TIMESTAMP special value. Then insert some values under two different time zones.

```
CREATE OR REPLACE TABLE TEST( COL1 TIMESTAMP DEFAULT TIMESTAMP,COL2 INT);
SET OPTION PUBLIC.time_zone='NewSouthWales';
INSERT INTO TEST(COL2) VALUES(1),(2);
SET OPTION PUBLIC.time_zone=;
INSERT INTO TEST(COL2) VALUES(3),(4);
SELECT * FROM TEST;
```

The first two rows are inserted using the current date and time in the NewSouthWales time zone and the last two rows are inserted using the current date and time in the server's time zone.

Related Information

CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
default_timestamp_increment Option
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
truncate_timestamp_values Option
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.1.8.21 USER Special Value

USER contains the user ID of the current connection.

Data Type

string

Remarks

USER can be used as a default value in columns with character data types. It is equivalent to CURRENT USER.

On UPDATE, columns with the USER default are not changed unless explicitly updated. Instead, the LAST USER default can be used to track updates by users.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Expressions in SQL Statements [page 34]
CURRENT USER Special Value [page 98]
LAST USER Special Value [page 102]

1.1.8.22 UTC TIMESTAMP Special Value

The UTC TIMESTAMP default value is used to record the Coordinated Universal Time (UTC) when a row in a table was last modified.

Data Type

TIMESTAMP WITH TIME ZONE

Remarks

The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

When a column is declared with DEFAULT UTC TIMESTAMP, a default value is provided for inserts, and the value is updated with the present UTC date and time of day whenever the row is updated.

Columns declared with DEFAULT UTC TIMESTAMP contain unique values so that applications can detect near-simultaneous updates to the same row. If the present UTC timestamp value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate UTC TIMESTAMP values with the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

> **i** Note
>
> DEFAULT UTC TIMESTAMP is set at both INSERT and UPDATE and DEFAULT CURRENT UTC TIMESTAMP is set at INSERT.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Expressions in SQL Statements [page 34]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
default_timestamp_increment Option
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]

TIMESTAMP Special Value [page 111]
TIMESTAMP WITH TIME ZONE Data Type [page 177]
truncate_timestamp_values Option
Time Zone Management
Creating Simulated Time Zones (SQL)

1.1.9 %TYPE and %ROWTYPE Attributes

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

Use the %TYPE and %ROWTYPE attributes when creating or declaring variables, converting values, creating or altering tables, and creating procedures, to define the data type(s) based on the data type of a column or row in a table, view, or cursor. The %TYPE attribute sets the data type to that of a column in the specified object, while the %ROWTYPE attribute sets the data types to those of a row in the specified object.

When %TYPE or %ROWTYPE is specified for a schema object, the database server derives the actual data type information from system tables. For example, if a %TYPE attribute specifies a table column, the data type is retrieved from the ISYSTABCOL system table.

Once the data types have been derived and the object (variable, column, and so on) is created, there is no further link or dependency to the object referenced in the %TYPE and %ROWTYPE attribute. However, in the case of procedures that use %TYPE and %ROWTYPE to define parameters and return types, the procedure can return different results if the underlying referenced objects change. This is because %TYPE and %ROWTYPE are evaluated when the procedure is executed, not when it is created.

Tables and Views

Specify the %TYPE attribute to set the data type of a column to the data type of a column in another table or view. For example:

- ```
  myColumnName other-table-name.column-name%TYPE
  ```

The second statement in the following example creates a table, myT2, and sets the data type of its column, myColumn, to the data type of the last_name column in myT1. Since additional attributes such as nullability are not applied, myT2.myColumn will not have the same NOT NULL restriction that myT1.last_name does.

```
CREATE TABLE myT1
( first_name   CHAR(20),
   last_name   VARCHAR NOT NULL );
CREATE TABLE myT2
( myColumn myT1.last_name%TYPE );
```
```

Procedures and Functions

Specify the %TYPE or %ROWTYPE attribute to set the data type(s) of the parameters to the data type(s) of a column or row in a specified table or view.

The following statement creates a procedure, DepartmentsCloseToCustomerLocation, and sets its IN parameter to the data type of the ID column in the Customers table:

```
CREATE OR REPLACE PROCEDURE DepartmentsCloseToCustomerLocation( IN customer_ID
Customers.ID%TYPE )
  BEGIN
    DECLARE cust_rec Customers%ROWTYPE;
    SELECT City, State, Country
    INTO cust_rec.City, cust_rec.State, cust_rec.Country
    FROM Customers
    WHERE ID = customer_ID;
    SELECT Employees.Surname, Employees.GivenName, Departments.DepartmentName
    FROM Employees JOIN Departments
    ON Departments.DepartmentHeadID = Employees.EmployeeID
    WHERE Employees.City = cust_rec.City
    AND Employees.State = cust_rec.State
    AND Employees.Country = cust_rec.Country;
  END;
CALL DepartmentsCloseToCustomerLocation(158);
```

The following statement creates a function called fullname and sets the data types of the firstname and lastname parameters to the data types of the Surname and Givenname column of the Employees table:

```
CREATE OR REPLACE FUNCTION fullname(
   firstname Employees.Surname%TYPE,
   lastname Employees.GivenName%TYPE )
RETURNS LONG VARCHAR
BEGIN
   RETURN ( firstname || ' ' || lastname );
END;
SELECT fullname ( Surname, GivenName )FROM GROUPO.Employees;
```

Casting and Converting Values

Specify the %TYPE attribute when to cast or convert a value to the data type of another database object.

The following statement casts a value to the data type defined for the BirthDate column (DATE data type) of the Employees table:

```
SELECT CAST ( '1966-10-30' AS Employees.BirthDate%TYPE );
```

Domains

Specify the %TYPE attribute to set the domain data type to the data type of a column in a specified table or view.

In the following example, the second two CREATE DOMAIN statements in the following set of statements create domains based on the data types of the Surname and GivenName columns of the Customers table.

```
CREATE DOMAIN identifier UNSIGNED INT
DEFAULT AUTOINCREMENT;
CREATE DOMAIN customers_surname Customers.Surname%TYPE;
CREATE DOMAIN customers_givenname Customers.GivenName%TYPE;

CREATE TABLE Customers3 (
   ID identifier PRIMARY KEY,
   SurName customers_surname,
   GivenName customers_givenname
);
```

Variables

Specify the %TYPE attribute to set the data type of a variable to the data type of a column in a specified table, view, or cursor. When %TYPE is used, only the data type is derived from the referenced object. Other column attributes such as default values, constraints, and whether NULLs are allowed, are not included and must be specified separately. Use the %TYPE attribute to declare a variable with the same type as column data when you want your application to be able to adjust to changes to an underlying table schema.

The following example creates a new variable, ProductID, and uses the %TYPE attribute to set its data type to the data type of the ID column in the Products table:

```
CREATE VARIABLE ProductID Products.ID%TYPE;
```

Specify the %ROWTYPE attribute to set the data type of a set of columns to the data types of a row in a specified table, view, or cursor. For example, use the %ROWTYPE attribute to define a variable that can store row or array values.

When %ROWTYPE is specified, other column attributes such as default values, constraints, and whether NULLs are allowed, are not included in the derivation.

The following example uses the %ROWTYPE attribute to create a variable, @a_product, and then inserts data from the Products table into the variable:

```
CREATE OR REPLACE PROCEDURE CheckStock (
    IN @id Products.ID%TYPE
)
   BEGIN
    DECLARE @a_product Products%ROWTYPE;
    SET (@a_product).ID = 200;
   END;
```

You can also use the %TYPE attribute to set the data type of a variable to type of another variable, as shown in the second DECLARE statement in this example:

```
DECLARE cust_rec Customers%ROWTYPE;
DECLARE cust_rec2 cust_rec%TYPE;
```

In this section:

%TYPE Attribute Syntax [page 118]

Sets the data type to that of a column in a specified object or variable when creating or declaring variables, or creating or altering tables, views, procedures, and functions. It can also be used for casting data from one type to another.

%ROWTYPE Attribute Syntax [page 120]
Sets the data type to the composite data type of a row in a specified table, view, table reference variable, or cursor.

1.1.9.1 %TYPE Attribute Syntax

Sets the data type to that of a column in a specified object or variable when creating or declaring variables, or creating or altering tables, views, procedures, and functions. It can also be used for casting data from one type to another.

Syntax

```
type-source%TYPE
| TYPE OF ( type-source )
type-source :
    [ owner. ]{ table-name
    | view-name }.column-name
    | variable-name
    | variable-name.field-name
```

Parameters

table-name

The name of a table.

view-name

The name of an enabled view (including materialized views). Materialized views must be initialized as well.

variable-name

The name of a variable.

column-name

The name of a column.

Remarks

When creating or altering procedures (parameters and return types), tables, views, and domains, an object that is referenced in a %TYPE specification must be a permanent object. A reference to a temporary object, such as a variable, cursor, or temporary table returns an error.

When %TYPE is specified in an IS OF search expression, a WITH `hint` expression in a FROM clause, or a CAST or CONVERT function, the referenced item must be a permanent object. Specifying a correlation name or a derived table returns an error.

When %TYPE is specified, other attributes, such as default values, constraints, and whether NULLs are allowed, are not part of the definition that is inherited and must be specified separately.

When defining or declaring a variable, if the identifier portion of `type-source` is one of the following, then the identifier portion must be quoted:

- IN
- OUT
- INOUT
- DYNAMIC
- SCROLL
- NO
- INSENSITIVE
- SENSITIVE
- TIMESTAMP
- a name that starts with #

For example, the statement below declares a variable called DYNAMIC. It then declares another variable called var1, and sets its data type to that of DYNAMIC (INT). Since DYNAMIC is one of the keywords that must be quoted, quotes are placed around it:

```
BEGIN
    DECLARE dynamic INT;
    DECLARE var1 "DYNAMIC"%TYPE;
    SET var1 = 1;
    MESSAGE var1;
END
```

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In addition to the following examples, there are examples in the documentation for the SQL statements and functions that support specifying the %TYPE attribute.

The following statement creates a procedure, DepartmentsCloseToCustomerLocation, and sets its IN parameter to the data type of the ID column in the Customers table using a %TYPE attribute:

```
CREATE OR REPLACE PROCEDURE DepartmentsCloseToCustomerLocation( IN customer_ID
Customers.ID%TYPE )
  BEGIN
    DECLARE cust_rec Customers%ROWTYPE;
    SELECT City, State, Country
    INTO cust_rec.City, cust_rec.State, cust_rec.Country
    FROM Customers
    WHERE ID = customer_ID;
    SELECT Employees.Surname, Employees.GivenName, Departments.DepartmentName
    FROM Employees JOIN Departments
    ON Departments.DepartmentHeadID = Employees.EmployeeID
    WHERE Employees.City = cust_rec.City
    AND Employees.State = cust_rec.State
    AND Employees.Country = cust_rec.Country;
  END;
CALL DepartmentsCloseToCustomerLocation(158);
```

The following statement casts a value to the data type defined for the BirthDate column (DATE data type) of the Employees table:

```
SELECT CAST ( '1966-10-30' AS Employees.BirthDate%TYPE );
```

Related Information

%TYPE and %ROWTYPE Attributes [page 115]
DECLARE Statement [page 1057]

1.1.9.2 %ROWTYPE Attribute Syntax

Sets the data type to the composite data type of a row in a specified table, view, table reference variable, or cursor.

Syntax

```
rowtype-source%ROWTYPE
| ROWTYPE OF ( rowtype-source )
rowtype-source :
    [ owner. ]{ table-name | view-name }
    | cursor-name
    | TABLE REF ( table-reference-variable )
```

Parameters

table-name

The name of a table.

When specifying `table-name`, the data type of the %ROWTYPE variable is comprised of the data types of the columns in `table-name`.

view-name

The name of an enabled view (including materialized views). Materialized views must be initialized as well.

When specifying `view-name`, the data type of the %ROWTYPE variable is comprised of the data types of the columns in `view-name`.

cursor-name

The name of a cursor.

When specifying `cursor-name`, the data type of the %ROWTYPE variable is comprised of the data types of the select items for the cursor.

table-reference-variable

The name of a table reference variable.

When specifying a table reference variable, the data type of the %ROWTYPE variable is comprised of the data types of the columns in the table referenced in `table-reference-variable`.

Remarks

When creating a %ROWTYPE variable, other attributes, such as default values, constraints, and whether NULLs are allowed, are not part of the definition that is inherited, and must be specified separately.

Restrictions when specifying %ROWTYPE with a table, view, or cursor reference:

When creating or altering procedures, views, and domains, an object referenced in a %ROWTYPE specification must be a permanent object. A reference to a temporary table returns an error.

If you declare a row variable and the argument to the %ROWTYPE construct is a cursor which is not yet opened, it is possible that the schema of the cursor will be different at open time if any of the underlying objects have changed. It is safer to declare row variables based on cursors that are already open.

When %ROWTYPE is specified in an IS OF search expression, a WITH `hint` expression in a FROM clause, or a CAST or CONVERT function, the referenced item must be a permanent object. Specifying a temporary table, correlation name, or a derived table, returns an error.

When `rowtype-source` references a cursor, the names of the items in the cursor (for example, column names) must be simple names, or an alias. If the select list item names in the cursor cannot be successfully derived, then an error is returned.

When defining or declaring a variable, if the identifier portion of `rowtype-source` is one of the following, then the identifier portion must be quoted:

- IN

- OUT
- INOUT
- DYNAMIC
- SCROLL
- NO
- INSENSITIVE
- SENSITIVE
- TIMESTAMP
- identifiers that start with #

Restrictions when specifying %ROWTYPE with a table reference variable (TABLE REF (table-reference-variable) %ROWTYPE):

Specifying %ROWTYPE with a table reference variable is not supported: when creating or altering procedures, views, and domains. Similarly, specifying %ROWTYPE with a table reference variable is not supported in an IS OF search expression, or in a WITH `hint` expression in a FROM clause, or in a CAST or CONVERT function.

When `rowtype-source` references a table reference variable, the table reference variable must already be initialized when the %ROWTYPE is processed. If `TABLE REF (table-reference-variable) %ROWTYPE` is used in a statement that is in a batch or procedure, statement must be nested inside another BEGIN...END block after the table reference variable has been assigned a value or passed as a parameter.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In addition to the following examples, there are examples in the documentation for the SQL statements and functions that support specifying the %ROWTYPE attribute.

The following example creates a new variable, ItemsForSale, and uses the %ROWTYPE attribute to set its data type to a composite data type comprised of the columns defined for the Products table:

```
CREATE VARIABLE ItemsForSale Products%ROWTYPE;
```

The following statement declares a variable, cust_rec, and sets its data type to the composite data type of a row in the Customers table:

```
CREATE OR REPLACE PROCEDURE DepartmentsCloseToCustomerLocation( IN customer_ID
Customers.ID%TYPE )
  BEGIN
    DECLARE cust_rec Customers%ROWTYPE;
    SELECT City, State, Country
    INTO cust_rec.City, cust_rec.State, cust_rec.Country
    FROM Customers
    WHERE ID = customer_ID;
    SELECT Employees.Surname, Employees.GivenName, Departments.DepartmentName
    FROM Employees JOIN Departments
    ON Departments.DepartmentHeadID = Employees.EmployeeID
    WHERE Employees.City = cust_rec.City
    AND Employees.State = cust_rec.State
    AND Employees.Country = cust_rec.Country;
  END;
CALL DepartmentsCloseToCustomerLocation(158);
```

Related Information

%TYPE and %ROWTYPE Attributes [page 115]
DECLARE Statement [page 1057]

1.1.10 SQL Variables

The supported variables can be grouped by scope: connection, database, and global.

When a variable is created, the initial value is set to NULL unless a default is specified. The value can subsequently be changed by using the SET statement, the UPDATE statement, or a SELECT statement with an INTO clause.

Variables are not affected by COMMIT or ROLLBACK statements.

Connection-scope variables

Connection-scope variables are set and used in the context of a connection. They are not available to other connections. There are two types of connection-scope variables: **connection-level** and **local** (also referred to as **declared**). You can also create connection-scope variables of type TABLE REF to hold references to tables; these are called table reference variables.

Connection-level variables

Connection-level variables are created by using the CREATE VARIABLE statement and are typically used to make values available to any procedure executed by the connection.

Connection-level variables persist only for the duration of the connection or until the variable is explicitly dropped by using the DROP VARIABLE statement

Local (declared) variables

Local variables are created by using the DECLARE statement inside of a BEGIN...END block, and are typically used to store and modify values within the same compound statement that the local variable is declared in. Local variable values are not available for use outside of the context of the BEGIN...END block.

Local variables persist only for the duration of the BEGIN...END block in which they are declared, and they can also be dropped.

Database-scope variables

Database-scope variables are used in the context of the database (instead of connection), and are a great way to share values across connections. Their intended use is to store small, infrequently changing, shared values. Storing large or frequently changing values may affect the performance of your application, and is not recommended. The initial values of database-scope variables persist after the database restarts (that is, changes to their initial value do not persist between database restarts). Database-scope variables can be used in the same manner as connection-scope and global variables, but they cannot be defined with the data type ROW, ARRAY, or TABLE REF.

Database-scope variables owned by users

When a database-scope variable is owned by a user, only that user can select from, and update, that variable, and can do so regardless of the connection.

Database-scope variables can also be owned by a role. However, the only access to a database-scope variable owned by a role is through the stored procedures, user-defined functions, and events owned by that role.

Database-scope variables owned by PUBLIC

Database variables owned by PUBLIC are available to all users and connections provided the users have the right system privileges.

Access to, and administration of, database-scope variables requires system privileges that vary depending on who owns the variable (self, another user, or PUBLIC). The following table summarizes the privileges required to access and administer database-scope variables:

Table 1: Privileges required for administering database-scope variables

Action	Owned by	Privilege Required
Create a database-scope variable	self	CREATE DATABASE VARIABLE or MANAGE ANY DATABASE VARIABLE
Create a database-scope variable	another user	MANAGE ANY DATABASE VARIABLE
Create a database-scope variable	PUBLIC	MANAGE ANY DATABASE VARIABLE
Update a database-scope variable	self	none required
Update a database-scope variable	another user	not allowed
Update a database-scope variable	PUBLIC	UPDATE PUBLIC DATABASE VARIABLE
Select from a database-scope variable	self	none
Select from a database-scope variable	another user	not allowed
Select from a database-scope variable	PUBLIC	SELECT PUBLIC DATABASE VARIABLE

Action	Owned by	Privilege Required
Drop a database-scope variable	self	none required
Drop a database-scope variable	another user	MANAGE ANY DATABASE VARIABLE
Drop a database-scope variable	PUBLIC	MANAGE ANY DATABASE VARIABLE

Global Variables

Global variables are used in the context of the database, but can only be set by the database server. Although you cannot directly set a global variable, some global variable values are indirectly set in response to user activity. For example, some global variables, such as @@identity, hold connection-specific information, while other variables, such as @@connections, have values that are common to all connections.

Global variables are visually distinguished from other variables by having two @ signs preceding their names. For example, @@error and @@rowcount are global variables.

Variables and Aliases with Identical Names

It is possible to have a statement that has aliases and variables with identical names. This is the sequence the database server follows when processing an identifier to help you know how the reference is resolved:

1. Match any aliases specified in the query SELECT list.
2. Match column names for any referenced tables.
3. Assume the name is a variable.

Standards

ANSI/ISO SQL Standard

Variables declared within SQL stored procedures or functions by using the DECLARE statement is supported in the ANSI/ISO SQL Standard as SQL Language Feature P002, "Computational completeness". CREATE VARIABLE, DROP VARIABLE, and global variables are not in the ANSI/ISO SQL Standard.

In this section:

@@identity Global Variable [page 126]
The @@identity variable holds the most recent value inserted by the current connection into an IDENTITY column, a DEFAULT AUTOINCREMENT column, or a DEFAULT GLOBAL AUTOINCREMENT column, or zero if the most recent insert was into a table that had no such column.

Related Information

CREATE VARIABLE Statement [page 1047]
DROP VARIABLE Statement [page 1145]
SET Statement [page 1373]
DECLARE Statement [page 1057]
UPDATE Statement [page 1463]
SYSDATABASEVARIABLE System View [page 1969]

1.1.10.1 @@identity Global Variable

The @@identity variable holds the most recent value inserted by the current connection into an IDENTITY column, a DEFAULT AUTOINCREMENT column, or a DEFAULT GLOBAL AUTOINCREMENT column, or zero if the most recent insert was into a table that had no such column.

The value of @@identity is connection specific. If a statement inserts multiple rows, @@identity reflects the IDENTITY value for the last row inserted. If the affected table does not contain an IDENTITY column, @@identity is set to zero.

The value of @@identity is not affected by the failure of an INSERT or SELECT INTO statement, or the rollback of the transaction that contained it. @@identity retains the last value inserted into an IDENTITY column, even if the statement that inserted it fails to commit.

@@identity and Triggers

When an insert causes referential integrity actions or fires a trigger, @@identity behaves like a stack. For example, if an insert into a table T1 (with an IDENTITY or AUTOINCREMENT column) fires a trigger that inserts a row into table T2 (also with an IDENTITY or AUTOINCREMENT column), then the value returned to the application or procedure which carried out the insert is the value inserted into T1. Within the trigger, @@identity has the T1 value before the insert into T2 and the T2 value after. The trigger can copy the values to local variables if it needs to access both.

Standards

ANSI/ISO SQL Standard

Global variables are not in the standard.

1.1.11 Comments

Comments are used to attach explanatory text to SQL statements or statement blocks. The database server does not execute comments.

The following comment indicators are supported:

-- (Double hyphen)

The database server ignores any remaining characters on the line. This is the ANSI/ISO SQL Standard comment indicator.

You can add and remove this comment indicator by selecting text and pressing Ctrl+Minus Sign in Interactive SQL or on the *SQL* tab of the *Procedures & Functions* window of *SQL Central*.

The SQL comment indicator is added to the beginning of each line of the selected text. If no text is selected, the comment indicator is added to the beginning of the current line.

// (Double slash)

The double slash has the same meaning as the double hyphen.

You can add and remove this comment indicator by selecting text and pressing Ctrl+Forward Slash in Interactive SQL or on the *SQL* tab of the *Procedures & Functions* window of *SQL Central*.

The SQL comment indicator is added to the beginning of each line of the selected text. If no text is selected, the comment indicator is added to the beginning of the current line.

/* ... */ (Slash-asterisk)

Any characters between the two comment markers are ignored. The two comment markers can be on the same or different lines. Comments indicated in this style can be nested, but nested comments must be balanced. Any comments made inside the comment block must not contain single instances of the starting comment marker. This style of commenting is also called **C-style comments**.

Example

The following example illustrates the use of double-hyphen comments:

```
CREATE FUNCTION fullname ( firstname CHAR(30),
         lastname CHAR(30))
RETURNS CHAR(61)
-- fullname concatenates the firstname and lastname
-- arguments with a single space between.
BEGIN
   DECLARE name CHAR(61);
   SET name = firstname || ' ' || lastname;
   RETURN ( name );
END;
```

The following example illustrates the use of C-style comments:

```
/* Lists the names and employee IDs of employees
   who work in the sales department. */
CREATE VIEW SalesEmployees AS
   SELECT EmployeeID, Surname, GivenName
   FROM GROUPO.Employees
```

```
    WHERE DepartmentID = 200;
```

Standards

ANSI/ISO SQL Standard

The use of double-minus signs for a comment is a core feature of the ANSI/ISO SQL Standard. The use of C-style, bracketed comments (/* ... */) is SQL Language Feature T351. Double-slash comments (//) are supported by the software but are not in the standard.

Related Information

Keyboard Shortcuts (Interactive SQL)

1.1.12 Named Parameters

Functions and procedures that are referenced from the CALL statement, the EXECUTE statement (Transact-SQL), the FROM clause of a DML statement, and the TRIGGER EVENT statement support positional parameters and named parameters. Named parameters support specifying any subset of the available parameters in any order.

Named parameters cannot be used with functions except in CALL statements. Named parameters cannot be used with built-in functions such as ABS, COMPRESS, DAYNAME.The following named parameter syntaxes are supported:

- `parameter-name = parameter-value`
- `parameter-name => parameter-value`

Example

The following example uses = to specify a named parameter:

```
CALL sa_conn_properties( connidparm = 1 );
```

This example uses named parameters when calling the system procedure sp_remote_exported_keys, and returns information about the foreign key relationships in the Employees table owned by HR on the remote server named RemoteSA:

```
CALL sp_remote_exported_keys(
     @server_name => 'RemoteSA',
     @table_owner => 'HR',
     @table_name => 'Employees' );
```

This example uses named parameters in the TRIGGER EVENT statement.

```
CREATE EVENT ev_TimePlace
HANDLER BEGIN
  MESSAGE 'ev_TimePlace - was triggered at ' ||  event_parameter( 'what_time' )
      || ' in ' || event_parameter( 'what_place' );
END;
TRIGGER EVENT ev_TimePlace( what_time => string( current timestamp ), what_place
=> 'Waterloo' );
```

Named parameters cannot be used with functions, except in CALL statements. The following is an example.

```
CREATE OR REPLACE FUNCTION PLUS( val1 INTEGER DEFAULT 0, val2 INTEGER DEFAULT 0 )
RETURNS INTEGER
BEGIN
    RETURN val1 + val2;
END
CREATE VARIABLE rslt INTEGER;
rslt = CALL PLUS( val1=1, val2=99 );
SELECT rslt;
```

Named parameters cannot be used with functions in general expressions. The following is an example of the syntax that must be used.

```
SELECT PLUS( 1, 99 );
```

Standards

ANSI/ISO SQL Standard

The = operator is not in the standard.

Related Information

CALL Statement [page 795]
EXECUTE IMMEDIATE Statement [SP] [page 1155]
EXECUTE Statement [ESQL] [page 1151]
EXECUTE Statement [T-SQL] [page 1153]
FROM Clause [page 1173]
TRIGGER EVENT Statement [page 1441]

1.2 SQL Data Types

There are many SQL data types supported by the software.

In this section:

Character Data Types [page 130]
Character data types store strings of letters, numbers, and other symbols.

Numeric Data Types [page 143]
Numeric data types store numerical data.

Money Data Types [page 156]
Money data types are used for storing monetary data.

Bit array Data Types [page 157]
A bit array is similar to a character string, except that the individual pieces are bit data (0s (zeros) and 1s (ones)) instead of characters. Typically, bit arrays are used to hold a string of Boolean values.

Date and Time Data Types [page 160]
Date values can be output in full century format, and the internal storage of dates always explicitly includes the century portion of a year value.

Binary Data Types [page 180]
Binary data types store binary data, including images and other types of information that are not interpreted by the database.

ROW and ARRAY Composite Data Types [page 185]
Composite data types are values that are comprised of zero or more elements, where each element has a value of a particular data type. Currently only ROW and ARRAY composite data types are supported.

TABLE REF Data Type [page 188]
The TABLE REF data type stores a reference to a base table, temporary table, or view. This data type is only for use with connection-scope variables.

Spatial Data Types [page 191]
Many spatial data types are supported. The documentation for these data types are located with the spatial SQL API documentation.

Domains [page 192]
Domains are aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are predefined, but you can add more of your own.

Data Type Comparisons [page 193]
When a comparison (such as =) is performed between arguments with different data types, one or more arguments must be converted so that the comparison operation is done using one data type.

Data Type Conversions [page 200]
Type conversions can happen automatically, or they can be explicitly requested using the CAST or CONVERT function. The following functions can also be used to force type conversions:

1.2.1 Character Data Types

Character data types store strings of letters, numbers, and other symbols.

There are classes of character data types and some domains defined using those types:

CHAR, VARCHAR, LONG VARCHAR

Character data stored in a single- or multibyte character set, often chosen to correspond most closely to the primary language or languages stored in the database.

NCHAR, NVARCHAR, LONG NVARCHAR

Character data stored in the UTF-8 Unicode encoding. All Unicode code points can be stored using these types, regardless of the primary language or languages stored in the database.

TEXT, UNIQUEIDENTIFIERSTR, XML

Domains based on other character data types.

UltraLite: UltraLite supports the CHAR, VARCHAR, and LONG VARCHAR data types, which are stored in a single- or multi- byte character set, and are often chosen to correspond most closely to the primary language or languages stored in the database.

Storage

All character data values are stored in the same manner. By default, values up to 128 bytes are stored in a single piece. Values longer than 128 bytes are stored with a 4-byte prefix kept locally on the database page and the full value stored in one or more other database pages. These default sizes are controlled by the INLINE and PREFIX clauses of the CREATE TABLE statement.

UltraLite: Fixed character types, such as VARCHAR, are embedded in the row whereas long character types, such as LONG VARCHAR, are stored separately. Consider your page size when creating a table with many columns of large fixed types. A full row must fit on a page, and fixed character column types are stored with a row. For example, a database created with a page size of 1000 cannot hold character values larger than 1000 because they cannot fit on the page.

In this section:

The XML data type stores character data of arbitrary length, and stores XML documents.

Related Information

CREATE TABLE Statement [page 1002]
string_rtruncation Option

1.2.1.1 CHAR Data Type

The CHAR data type stores character data, up to 32767 bytes.

Syntax

```
CHAR [ ( max-length [ CHAR | CHARACTER ] ) ]
```

UltraLite:

```
CHAR [ ( max-length) ]
```

Parameters

max-length

The maximum length of the string. If byte-length semantics are used (CHAR or CHARACTER is not specified as part of the length), then the length is in bytes, and the length must be in the range 1 to 32767. If the length is not specified, then it is 1.

If character-length semantics are used (CHAR or CHARACTER is specified as part of the length), then the length is in characters, and you must specify `max-length`. `max-length` can be a maximum of 32767 characters.

Remarks

Multibyte characters can be stored as CHAR, but the declared length refers to bytes, not characters, unless character-length semantics are used.

CHAR can also be specified as CHARACTER. Regardless of which syntax is used, the data type is described as CHAR.

CHAR is semantically equivalent to VARCHAR, although they are different types. CHAR is a variable-length type. In other relational database management systems, CHAR is a fixed-length type, and data is padded with blanks to `max-length` bytes of storage. SQL Anywhere does not blank-pad stored character data.

How CHAR columns are described depends on the client interface, the character sets used, and if character-length semantics are used. For example, in Embedded SQL the described length is the maximum number of bytes in the client character set. If the described length would be more than 32767 bytes, the column is described as type DT_LONGVARCHAR. The following table shows some Embedded SQL examples and the results returned when a DESCRIBE is performed:

Type being described	Database character set	Client character set	Result of DESCRIBE
CHAR(10)	Windows-1252	Windows-1252	DT_FIXCHAR length 10
CHAR(10)	UTF-8	UTF-8	DT_FIXCHAR length 10
CHAR(10)	Windows-1252	UTF-8	DT_FIXCHAR length 30
CHAR(20000)	Windows-31J	UTF-8	DT_LONGVARCHAR
CHAR(10 CHAR)	Windows-1252	Windows-1252	DT_FIXCHAR length 10
CHAR(10 CHAR)	UTF-8	UTF-8	DT_FIXCHAR length 40

For ODBC, CHAR is described as either SQL_CHAR or SQL_VARCHAR depending on the odbc_distinguish_char_and_varchar option.

UltraLite: CHAR is a domain, implemented as VARCHAR.

Standards

ANSI/ISO SQL Standard

Compatible with the ANSI/ISO SQL Standard. In the standard, character-length semantics are the default, whereas in the software, byte-length semantics are the default. There are minor inconsistencies with the SQL standard due to case-insensitive collation support and the software's support for blank-padding.

The ANSI/ISO SQL Standard supports explicit character- or byte-length semantics as SQL Language Feature T061.

Related Information

odbc_distinguish_char_and_varchar Option
VARCHAR Data Type [page 140]
LONG VARCHAR Data Type [page 135]
NCHAR Data Type [page 136]

1.2.1.2 LONG NVARCHAR Data Type

The LONG NVARCHAR data type stores Unicode character data of arbitrary length.

Syntax

LONG NVARCHAR

Remarks

The maximum size in bytes is 2 GB minus 1 (2^{31} - 1) or 2 147 483 647.

Characters are stored in UTF-8. Each character requires from one to four bytes. If all the characters can be represented as single-byte UTF-8, the maximum number of characters that can be stored in a LONG NVARCHAR is 2 147 483 647 characters. If all the characters can be represented as double-byte UTF-8, the maximum number of characters that can be stored in a LONG NVARCHAR is 1 073 741 823 characters.

When an Embedded SQL client performs a DESCRIBE on a LONG NVARCHAR column, the data type returned is either DT_LONGVARCHAR or DT_LONGNVARCHAR, depending on whether the db_change_nchar_charset function has been called.

For ODBC, a LONG NVARCHAR expression is described as SQL_WLONGVARCHAR.

Standards

ANSI/ISO SQL Standard

The use of LONG NVARCHAR to declare a national character string is not in the standard.

Related Information

db_change_nchar_charset Function
NCHAR Data Type [page 136]
NVARCHAR Data Type [page 138]
LONG VARCHAR Data Type [page 135]

1.2.1.3 LONG VARCHAR Data Type

The LONG VARCHAR data type stores character data of arbitrary length.

Syntax

LONG VARCHAR

Remarks

The maximum size in bytes is 2 GB minus 1 (2^{31} - 1) or 2 147 483 647.

Multibyte characters can be stored as LONG VARCHAR, but the length is in bytes, not characters.

UltraLite:

- You can cast strings to/from LONG VARCHAR data.
- LONG VARCHAR data cannot be concatenated.
- LONG VARCHAR columns can be included in the result set of a SELECT query.
- Indexes cannot be created on a LONG VARCHAR type.
- A LONG VARCHAR type can only be used in the LENGTH and CAST functions.
- Conditions in SQL statements, such as in the WHERE clause, cannot operate on LONG VARCHAR columns.
- Only INSERT, UPDATE, and DELETE operations are allowed on LONG VARCHAR column.

Standards

ANSI/ISO SQL Standard

Large object support is SQL Language Feature T041.

Related Information

CHAR Data Type [page 132]
VARCHAR Data Type [page 140]
LONG NVARCHAR Data Type [page 134]

1.2.1.4 NCHAR Data Type

The NCHAR data type stores Unicode character data, up to 32767 characters.

Syntax

```
NCHAR [ ( max-length ) ]
```

Parameters

max-length

The maximum length of the string, in characters. The length must be in the range 1 to 32767. If the length is not specified, then it is 1.

Remarks

Characters are stored using UTF-8 encoding. The maximum number of bytes of storage required is four multiplied by `max-length`. However, the actual number of bytes of storage required is usually much less.

For example, the encoding of the character Yee from the Deseret alphabet (U+10437) in UTF-8 requires 4 bytes. The following SQL query displays the character Yee when using Interactive SQL.

```
SELECT CAST(0xF09090B7 as NCHAR(1));
```

NCHAR can also be specified as NATIONAL CHAR or NATIONAL CHARACTER. Regardless of which syntax is used, the data type is described as NCHAR.

NCHAR is semantically equivalent to NVARCHAR, although they are different types. Note that NCHAR is treated as a variable-length type and columns are not blank-padded when stored.

When an Embedded SQL client performs a DESCRIBE on an NCHAR column, the data type returned is either DT_FIXCHAR or DT_NFIXCHAR, depending on whether the db_change_nchar_charset function has been called.

Also, when an Embedded SQL client performs a DESCRIBE on an NCHAR column, the length returned is the maximum byte length in the client NCHAR character set. For example, for an Embedded SQL client using the Western European character set cp1252 as the NCHAR character set, an NCHAR(10) column is described as type DT_NFIXCHAR of length 10 (10 characters multiplied by a maximum one byte per character). For an Embedded SQL client using the Japanese character set cp932, the same column is described as type DT_NFIXCHAR of length 20 (10 characters multiplied by a maximum two bytes per character). If the described length would return more then 32767 bytes, the column is described as type DT_LONGNVARCHAR.

For ODBC, if the byte length (octet length) is less than 32768, NCHAR is described as SQL_WCHAR; otherwise, it is described as SQL_WLONGVARCHAR. For example, NCHAR(8191) requires a maximum 32764 bytes of storage so it is described as SQL_WCHAR with octet length 32764. However, NCHAR(8192) requires a maximum 32768 bytes of storage so it is described as SQL_WLONGVARCHAR with octet length 2147483647.

Standards

ANSI/ISO SQL Standard

National character support is Feature F421.

Related Information

db_change_nchar_charset Function
CHAR Data Type [page 132]
NVARCHAR Data Type [page 138]
LONG NVARCHAR Data Type [page 134]

1.2.1.5 NTEXT Data Type

The NTEXT data type stores Unicode character data of arbitrary length.

Syntax

NTEXT

Remarks

NTEXT is a domain, implemented as a LONG NVARCHAR.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

LONG NVARCHAR Data Type [page 134]
TEXT Data Type [page 139]

1.2.1.6 NVARCHAR Data Type

The NVARCHAR data type stores Unicode character data, up to 32767 characters.

Syntax

```
NVARCHAR [ ( max-length ) ]
```

Parameters

max-length

The maximum length of the string, in characters. The length must be in the range 1 to 32767. If the length is not specified, then it is 1.

Remarks

Characters are stored in UTF-8 encoding. The maximum number of bytes of storage required is four multiplied by `max-length`, although the actual storage required is usually much less.

For example, the encoding of the character Yee from the Deseret alphabet (U+10437) in UTF-8 requires 4 bytes. The following SQL query displays the character Yee when using Interactive SQL.

```
SELECT CAST(0xF09090B7 as NVARCHAR(1));
```

NVARCHAR can also be specified as NCHAR VARYING, NATIONAL CHAR VARYING, or NATIONAL CHARACTER VARYING. Regardless of which syntax is used, the data type is described as NVARCHAR.

When an Embedded SQL client performs a DESCRIBE on a NVARCHAR column, the data type returned is either DT_VARCHAR or DT_NVARCHAR, depending on whether the db_change_nchar_charset function has been called.

Also, when an Embedded SQL client performs a DESCRIBE on an NVARCHAR column, the length returned is the maximum byte length in the client NCHAR character set. For example, for an Embedded SQL client using the Western European character set cp1252 as the NCHAR character set, an NVARCHAR(10) column is described as type DT_NVARCHAR of length 10 (10 characters multiplied by a maximum of one byte per character). For an Embedded SQL client using the Japanese character set cp932, the same column is described as type DT_NVARCHAR of length 20 (10 characters multiplied by a maximum two bytes per character). If the describe length would return more than 32767 bytes, the column is described as type DT_LONGNVARCHAR.

For ODBC, if the byte length (octet length) is less than 32768, NVARCHAR is described as SQL_WVARCHAR; otherwise, it is described as SQL_WLONGVARCHAR. For example, NVARCHAR(8191) requires a maximum 32764 bytes of storage so it is described as SQL_WVARCHAR with octet length 32764. However, NVARCHAR(8192) requires a maximum 32768 bytes of storage so it is described as SQL_WLONGVARCHAR with octet length 2147483647.

Standards

ANSI/ISO SQL Standard

National character support is SQL Language Feature F421.

Related Information

db_change_nchar_charset Function
NCHAR Data Type [page 136]
LONG NVARCHAR Data Type [page 134]
VARCHAR Data Type [page 140]

1.2.1.7 TEXT Data Type

The TEXT data type stores character data of arbitrary length.

Syntax

```
TEXT
```

Remarks

TEXT is a domain, implemented as a LONG VARCHAR.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

LONG VARCHAR Data Type [page 135]
NTEXT Data Type [page 137]

1.2.1.8 UNIQUEIDENTIFIERSTR Data Type

UNIQUEIDENTIFIERSTR is a domain, implemented as CHAR(36).

Syntax

```
UNIQUEIDENTIFIERSTR
```

Remarks

Used for remote data access, when mapping Microsoft SQL Server uniqueidentifier columns.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Server Class MSSODBC
STRTOUUID Function [String] [page 575]

1.2.1.9 VARCHAR Data Type

The VARCHAR data type stores character data, up to 32767 bytes.

Syntax

```
VARCHAR [ ( max-length [ CHAR | CHARACTER ] ) ]
```

UltraLite:

```
VARCHAR [ ( max-length) ]
```

Parameters

max-length

The maximum length of the string. This default value is 1.

If byte-length semantics are used (CHAR or CHARACTER is not specified as part of the length), then the length is in bytes, and the length must be in the range of 1 to 32767.

If character-length semantics are used (CHAR or CHARACTER is specified as part of the length), then the length is in characters, and you must specify `max-length`. `max-length` can be a maximum of 32767 characters.

UltraLite: UltraLite databases only support byte-length semantics. A non-English character can require up to 3 bytes of storage.

Remarks

Multibyte characters can be stored as VARCHAR, but the declared length refers to bytes, not characters, unless character-length semantics are used.

UltraLite: UltraLite compacts data as much as possible. When a VARCHAR value does not require the number of bytes specified by `max-length`, then only the number of bytes needed to store the value is used. When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.

> ⚠ Caution
>
> **UltraLite:**
>
> Although it is possible to create a table with a VARCHAR column where the `max-length` exceeds the page size, an error occurs if you insert a value with a length exceeding that page size.

VARCHAR can also be specified as CHAR VARYING or CHARACTER VARYING. Regardless of which syntax is used, the data type is described as VARCHAR.

VARCHAR is semantically equivalent to CHAR, although they are different types. In SQL Anywhere, CHAR is a variable-length type. In other relational database management systems, CHAR is a fixed-length type, and data is padded with blanks to `max-length` bytes of storage. SQL Anywhere does not blank-pad stored character data.

How VARCHAR columns are described depends on the client interface, the character sets used, and if character-length semantics are used. For example, in Embedded SQL the described length is the maximum number of bytes in the client character set. If the described length would be more than 32767 bytes, the column is described as type DT_LONGVARCHAR. The following table shows some Embedded SQL examples and the results returned when a DESCRIBE is performed:

Type being described	Database character set	Client character set	Result of DESCRIBE
VARCHAR(10)	Windows-1252	Windows-1252	DT_VARCHAR length 10
VARCHAR(10)	UTF-8	UTF-8	DT_VARCHAR length 10

Type being described	Database character set	Client character set	Result of DESCRIBE
VARCHAR(10)	Windows-1252	UTF-8	DT_VARCHAR length 30
VARCHAR(20000)	Windows-31J	UTF-8	DT_LONGVARCHAR
VARCHAR(10 CHAR)	Windows-1252	Windows-1252	DT_VARCHAR length 10
VARCHAR(10 CHAR)	UTF-8	UTF-8	DT_VARCHAR length 40

For ODBC, VARCHAR is described as SQL_VARCHAR.

Standards

ANSI/ISO SQL Standard

Compatible with the ANSI/ISO SQL Standard. In the standard, character-length semantics are the default, whereas in the software, byte-length semantics are the default. There are minor inconsistencies with the SQL standard due to case-insensitive collation support and support for blank-padding by the software.

The ANSI/ISO SQL Standard supports explicit character- or byte-length semantics as SQL Language Feature T061.

Related Information

CHAR Data Type [page 132]
LONG VARCHAR Data Type [page 135]
NVARCHAR Data Type [page 138]

1.2.1.10 XML Data Type

The XML data type stores character data of arbitrary length, and stores XML documents.

Syntax

XML

Remarks

The maximum size is 2 GB minus 1 byte (2^{31} - 1).

Data of type XML is not quoted when generating element content from relational data.

You can cast between the XML data type and any other data type that can be cast to or from a string. There is no checking that the string is well-formed when it is cast to XML.

When an Embedded SQL client application performs a DESCRIBE on an XML column, it is described as LONG VARCHAR.

Standards

ANSI/ISO SQL Standard

The XML data type is SQL Language Feature X010.

Related Information

XML in the Database
Storage of XML Documents in Relational Databases

1.2.2 Numeric Data Types

Numeric data types store numerical data.

The NUMERIC and DECIMAL data types, and the various INTEGER data types, are sometimes called **exact** numeric data types, in contrast to the **approximate** numeric data types FLOAT, DOUBLE, and REAL.

The exact numeric data types are those for which precision and scale values can be specified, while approximate numeric data types are stored in a predefined manner. *Only exact numeric data is guaranteed accurate to the least significant digit specified after an arithmetic operation*.

Data type lengths and precision of less than one are not allowed.

Compatibility

Be careful when using default precision and scale settings for NUMERIC and DECIMAL data types because these settings could be different in other database solutions. The default precision is 30 and the default scale is 6.

The FLOAT (`p`) data type is a synonym for REAL or DOUBLE, depending on the value of `p`. For SQL Anywhere, the cutoff is platform-dependent, but on all platforms the cutoff value is greater than 15.

Only the NUMERIC data type with scale = 0 can be used for the Transact-SQL identity column. Avoid default precision and scale settings for NUMERIC and DECIMAL data types, because these are different between SQL Anywhere and Adaptive Server Enterprise. In SQL Anywhere, the default precision is 30 and the default scale is 6. In Adaptive Server Enterprise, the default precision is 18 and the default scale is 0.

In this section:

Related Information

precision Option
scale Option

1.2.2.1 BIGINT Data Type

The BIGINT data type stores BIGINTs, which are integers requiring 8 bytes of storage.

Syntax

```
[ UNSIGNED ] BIGINT
```

Remarks

The BIGINT data type is an exact numeric data type: its accuracy is preserved after arithmetic operations.

A BIGINT value requires 8 bytes of storage.

The range for BIGINT values is -2^{63} to $2^{63} - 1$, or -9223372036854775808 to 9223372036854775807.

The range for UNSIGNED BIGINT values is 0 to $2^{64} - 1$, or 0 to 18446744073709551615.

By default, the data type is signed.

When converting a string to a BIGINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

Standards

ANSI/ISO SQL Standard

SQL Language Feature T071.

MySQL

The UNSIGNED keyword may follow BIGINT.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
BIT Data Type [page 145]
INTEGER Data Type [page 150]
SMALLINT Data Type [page 154]
TINYINT Data Type [page 155]

1.2.2.2 BIT Data Type

The BIT data type stores a bit (0 or 1).

Syntax

BIT

Remarks

BIT is an integer type that can store the values 0 or 1.

By default, the BIT data type does not allow NULL.

When converting a string to a BIT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is not 0 or 1.

A BIT value requires 1 byte of storage.

UltraLite: A BIT value requires 1 bit of storage.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
BIGINT Data Type [page 144]
INTEGER Data Type [page 150]
SMALLINT Data Type [page 154]
TINYINT Data Type [page 155]

1.2.2.3 DECIMAL Data Type

The DECIMAL data type is a decimal number with `precision` total digits and with `scale` digits after the decimal point.

Syntax

```
DECIMAL [ ( precision [ , scale ] ) ]
```

Parameters

precision

An integer expression between 1 and 127, inclusive, that specifies the number of digits in the expression. The default setting is 30.

scale

An integer expression between 0 and 127, inclusive, that specifies the number of digits after the decimal point. The scale value should always be less than, or equal to, the precision value.

If precision and scale are both omitted, the default scale is 6. If precision is specified but scale is omitted, the default scale is 0.

UltraLite: Change the defaults by setting the appropriate creation parameter.

Remarks

The DECIMAL data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations.

The number of bytes required to store a decimal number can be estimated as

```
2 + INT(((precision - scale) + 1) / 2) + INT((scale + 1) / 2);
```

The INT function takes the integer portion of its argument. The storage is based on the value being stored, not on the maximum precision and scale allowed in the column.

If you are using a precision of 20 or less and a scale of 0, it may be possible to use one of the integer data types (BIGINT, INTEGER, SMALLINT, or TINYINT) instead. Integer values require less storage space than NUMERIC and DECIMAL values with a similar number of significant digits. Operations on integer values, such as fetching or inserting, and arithmetic operators, typically perform better than operations on NUMERIC and DECIMAL values.

> **i** Note
>
> If you create a column or variable of a DECIMAL data type with a precision or scale that exceeds the precision and scale settings for the database, values are truncated to the database settings. So, if you notice truncated values in a column or variable defined as DECIMAL, check that precision and scale do not exceed the database option settings.

DECIMAL can also be specified as DEC. Regardless of which syntax is used, the data type is described as DECIMAL. DECIMAL is semantically equivalent to NUMERIC.

Standards

ANSI/ISO SQL Standard

Core Feature.

Example

```
DECLARE d1 DECIMAL; // the default scale is 6
DECLARE d2 DECIMAL ( 20 ); // the default scale is 0
```

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
FLOAT Data Type [page 149]
REAL Data Type [page 153]
DOUBLE Data Type [page 148]
NUMERIC Data Type [page 151]
precision Option
scale Option

1.2.2.4 DOUBLE Data Type

The DOUBLE data type stores double-precision floating-point numbers.

Syntax

```
DOUBLE
```

Remarks

The DOUBLE data type is an approximate numeric data type and subject to rounding errors after arithmetic operations. The approximate nature of DOUBLE values means that queries using equalities should generally be avoided when comparing DOUBLE values.

DOUBLE values require 8 bytes of storage.

The range of values is -1.79769313486231e+308 to 1.79769313486231e+308, with numbers close to zero as small as 2.22507385850721e-308. Values held as DOUBLE are accurate to 15 significant digits, but may be subject to rounding errors beyond the fifteenth digit.

Standards

ANSI/ISO SQL Standard

Core Feature

Related Information

Numeric Functions [page 219]
Numeric Set Conversions [page 204]
Aggregate Functions [page 208]
FLOAT Data Type [page 149]
REAL Data Type [page 153]
DECIMAL Data Type [page 146]
NUMERIC Data Type [page 151]

1.2.2.5 FLOAT Data Type

The FLOAT data type stores a floating-point number, which can be single or double precision.

Syntax

```
FLOAT [ ( precision ) ]
```

Parameters

precision

An integer expression that specifies the number of bits in the mantissa, the decimal part of a logarithm. For example, in the number 5.63428, the mantissa is 0.63428. The IEEE standard 754 floating-point precision is as follows:

Supplied precision value	Decimal precision	Equivalent SQL data type	Storage size
1-24	7 decimal digits	REAL	4 bytes
25-53	15 decimal digits	DOUBLE	8 bytes

Remarks

When a column is created using the FLOAT (`precision`) data type, columns on all platforms are guaranteed to hold the values to at least the specified minimum precision. REAL and DOUBLE do not guarantee a platform-independent minimum precision.

If `precision` is not supplied, the FLOAT data type is a single-precision floating-point number, equivalent to the REAL data type, and requires 4 bytes of storage.

If `precision` is supplied, the FLOAT data type is either single or double precision, depending on the value of precision specified. The cutoff between REAL and DOUBLE is platform-dependent. Single-precision FLOAT values require 4 bytes of storage, and double-precision FLOAT values require 8 bytes.

The FLOAT data type is an approximate numeric data type. It is subject to rounding errors after arithmetic operations. The approximate nature of FLOAT values means that queries using equalities should be avoided when comparing FLOAT values.

Standards

ANSI/ISO SQL Standard

Core Feature.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
DOUBLE Data Type [page 148]
REAL Data Type [page 153]
DECIMAL Data Type [page 146]
NUMERIC Data Type [page 151]

1.2.2.6 INTEGER Data Type

The INTEGER data type stores integers that require 4 bytes of storage.

Syntax

```
[ UNSIGNED ] INTEGER
```

Remarks

The INTEGER data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

If you specify UNSIGNED, the integer can never be assigned a negative number. By default, the data type is signed.

The range for INTEGER values is -2^{31} to $2^{31} - 1$, or -2147483648 to 2147483647.

The range for UNSIGNED INTEGER values is 0 to $2^{32} - 1$, or 0 to 4294967295.

When converting a string to an INTEGER, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

Standards

ANSI/ISO SQL Standard

Core Feature. However, the UNSIGNED keyword is not in the standard.

MySQL

The UNSIGNED keyword may follow INTEGER.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
BIGINT Data Type [page 144]
BIT Data Type [page 145]
SMALLINT Data Type [page 154]
TINYINT Data Type [page 155]

1.2.2.7 NUMERIC Data Type

The NUMERIC data type stores decimal numbers with `precision` total digits and with `scale` digits after the decimal point.

Syntax

```
NUMERIC [ ( precision [ , scale ] ) ]
```

Parameters

precision

An integer expression between 1 and 127, inclusive, that specifies the number of digits in the expression. The default setting is 30.

scale

An integer expression between 0 and 127, inclusive, that specifies the number of digits after the decimal point. The scale value should always be less than or equal to the precision value. The default setting is 6.

Remarks

The NUMERIC data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations.

UltraLite: NUMERIC is a domain, implemented as DECIMAL.

The number of bytes required to store a decimal number can be estimated as

```
2 + INT( (BEFORE+1)/2 ) + INT( (AFTER+1)/2 );
```

The INT function takes the integer portion of its argument, and BEFORE and AFTER are the number of significant digits before and after the decimal point. The storage is based on the value being stored, not on the maximum precision and scale allowed in the column.

If you are using a precision of 20 or less and a scale of 0, it may be possible to use one of the integer data types (BIGINT, INTEGER, SMALLINT, or TINYINT) instead. Integer values require less storage space than NUMERIC and DECIMAL values with a similar number of significant digits. Operations on integer values, such as fetching or inserting, and arithmetic operators, typically perform better than operations on NUMERIC and DECIMAL values.

NUMERIC is semantically equivalent to DECIMAL.

> **i** Note
>
> If you create a column or variable of a NUMERIC data type with a precision or scale that exceeds the precision and scale settings for the database, values are truncated to the database settings. So, if you notice truncated values in a column or variable defined as NUMERIC, check that precision and scale do not exceed the database option settings.

Standards

ANSI/ISO SQL Standard

Compatible with ANSI/ISO SQL Standard if the scale option is set to zero.

Related Information

Numeric Functions [page 219]

Aggregate Functions [page 208]
Numeric Set Conversions [page 204]
FLOAT Data Type [page 149]
REAL Data Type [page 153]
DOUBLE Data Type [page 148]
DECIMAL Data Type [page 146]
precision Option
scale Option

1.2.2.8 REAL Data Type

The REAL data type stores single-precision floating-point numbers stored in 4 bytes.

Syntax

REAL

Remarks

The REAL data type is an approximate numeric data type and subject to rounding errors after arithmetic operations. The approximate nature of REAL values means that queries using equalities should generally be avoided when comparing REAL values.

REAL values require 4 bytes of storage.

The range of values is -3.402823e+38 to 3.402823e+38, with numbers close to zero as small as 1.175494351e-38. Values held as REAL are accurate to 7 significant digits, but may be subject to rounding error beyond the sixth digit.

Standards

ANSI/ISO SQL Standard

Core Feature.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
DOUBLE Data Type [page 148]

FLOAT Data Type [page 149]
DECIMAL Data Type [page 146]
NUMERIC Data Type [page 151]

1.2.2.9 SMALLINT Data Type

The SMALLINT data type stores integers that require 2 bytes of storage.

Syntax

```
[ UNSIGNED ] SMALLINT
```

Remarks

The SMALLINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations. It requires 2 bytes of storage.

The range for SMALLINT values is -2^{15} to $2^{15} - 1$, or -32768 to 32767.

The range for UNSIGNED SMALLINT values is 0 to $2^{16} - 1$, or 0 to 65535.

When converting a string to a SMALLINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

Standards

ANSI/ISO SQL Standard

Compatible with the standard. However, the UNSIGNED keyword is not in the standard.

MySQL

The UNSIGNED keyword may follow SMALLINT.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
BIGINT Data Type [page 144]

BIT Data Type [page 145]
INTEGER Data Type [page 150]
TINYINT Data Type [page 155]

1.2.2.10 TINYINT Data Type

The TINYINT data type stores unsigned integers requiring 1 byte of storage.

Syntax

TINYINT

Remarks

The TINYINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

The range for TINYINT values is 0 to 2^8 - 1, or 0 to 255.

When converting a string to a TINYINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

In Embedded SQL, TINYINT columns should not be fetched into variables defined as CHAR or UNSIGNED CHAR, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program. Instead, TINYINT columns should be fetched into 2-byte or 4-byte integer columns. To send a TINYINT value to a database from an application written in C, the type of the C variable should be INTEGER.

UltraLite: In Embedded SQL, TINYINT columns should not be fetched into variables defined as CHAR, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program. Instead, TINYINT columns should be fetched into 2-byte or 4-byte integer columns. To send a TINYINT value to a database from an application written in C, the type of the C variable should be INTEGER.

Standards

ANSI/ISO SQL Standard

Not in the standard.

MySQL

The UNSIGNED keyword may precede or follow TINYINT, but the UNSIGNED modifier has no effect as the type is always unsigned.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
BIGINT Data Type [page 144]
BIT Data Type [page 145]
INTEGER Data Type [page 150]
SMALLINT Data Type [page 154]

1.2.3 Money Data Types

Money data types are used for storing monetary data.

In this section:

MONEY Data Type [page 156]
The MONEY data type stores monetary data.

SMALLMONEY Data Type [page 157]
The SMALLMONEY data type stores monetary data that is less than one million currency units.

1.2.3.1 MONEY Data Type

The MONEY data type stores monetary data.

Syntax

MONEY

Remarks

MONEY is a domain, implemented as NUMERIC(19,4).

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
SMALLMONEY Data Type [page 157]

1.2.3.2 SMALLMONEY Data Type

The SMALLMONEY data type stores monetary data that is less than one million currency units.

Syntax

SMALLMONEY

Remarks

SMALLMONEY is a domain, implemented as NUMERIC(10,4).

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Numeric Functions [page 219]
Aggregate Functions [page 208]
MONEY Data Type [page 156]

1.2.4 Bit array Data Types

A bit array is similar to a character string, except that the individual pieces are bit data (0s (zeros) and 1s (ones)) instead of characters. Typically, bit arrays are used to hold a string of Boolean values.

The bit array data types supported include VARBIT and LONG VARBIT.

In this section:

1.2.4.1 LONG VARBIT Data Type

The LONG VARBIT data type stores arbitrary length bit arrays.

Syntax

LONG VARBIT

Remarks

Used to store arbitrary length array of bits (1s and 0s), or bit arrays longer than 32767 bits.

LONG VARBIT can also be specified as LONG BIT VARYING. Regardless of which syntax is used, the data type is described as LONG VARBIT.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Bit Array Conversions [page 202]
Bit Array Functions [page 210]
Aggregate Functions [page 208]
BIT Data Type [page 145]
VARBIT Data Type [page 159]

1.2.4.2 VARBIT Data Type

The VARBIT data type is used for storing bit arrays that are at most 32767 bits in length.

Syntax

```
VARBIT [ (max-length ) ]
```

Parameters

max-length

The maximum length of the bit array, in bits. The length must be in the range 1 to 32767. If the length is not specified, then it is 1.

Remarks

VARBIT can also be specified as BIT VARYING. Regardless of which syntax is used, the data type is described as VARBIT.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Bit Array Conversions [page 202]
Bit Array Functions [page 210]
Aggregate Functions [page 208]
BIT Data Type [page 145]
LONG VARBIT Data Type [page 158]
Bitwise Operators [page 33]

1.2.5 Date and Time Data Types

Date values can be output in full century format, and the internal storage of dates always explicitly includes the century portion of a year value.

Correct values are always returned for any legal arithmetic and logical operations on dates, regardless of whether the calculated values span different centuries.

In this section:

1.2.5.1 How Date and Time of Day are Stored

Date and time of day are stored in the database using an appropriate data type.

Data type	Contains	Stored in	Range of possible values
DATE	Calendar date (year, month, day)	4 bytes	Dates from 0001-01-01 to 9999-12-31.
TIME	Time of day (hour, minute, second, and fraction of a second accurate to 6 decimal places)	8 bytes	Times from 00:00:00.000000 to 24:00:00.000000.
TIMESTAMP	Calendar date and time of day (year, month, day, hour, minute, second, and fraction of a second accurate to 6 decimal places)	8 bytes	Dates from 0001-01-01 to 9999-12-31.
TIMESTAMP WITH TIME ZONE	Calendar date, time of day, and time zone offset (year, month, day, hour, minute, second, fraction of a second accurate to 6 decimal places, and time zone offset in hours and minutes)	10 bytes	Dates from 0001-01-01 to 9999-12-31 (the hours and minutes portion of a TIMESTAMP WITH TIME ZONE is dropped after 7911-01-01 00:00:00 but seconds and fractional seconds are preserved). Zone offset from -14:59 to +14:59.

1.2.5.2 Ways to Send Dates and Times to the Database

The date and time of day can be sent to the database as a string using any interface. It can also be sent using ODBC or OLE DB, as a binary value (using an ODBC TIMESTAMP_STRUCT structure, for example), or using Embedded SQL, as a SQLDATETIME structure

The date and time of day with a time zone offset can be sent to the database as a string only.

In this section:

Date Formats [page 162]
When a date is sent to the database as a string (for the DATE data type) or as part of a string (for the TIMESTAMP or TIMESTAMP WITH TIME ZONE data types), the string can be specified in several different ways including that described by the ISO 8601 international standard.

Time Formats [page 164]
The time of day can be specified in the ISO 8601 format, using the 24-hour timekeeping system.

Date with Time Formats [page 165]
ISO 8601 permits the date and time of day to be combined using a space character or the letter T.

Time Zone Formats [page 166]
ISO 8601 also permits the addition of a time zone offset to a date and time of day string.

Related Information

Comparisons of Dates and Times [page 197]
DATE Data Type [page 168]
date_format Option
nearest_century Option
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]

1.2.5.2.1 Date Formats

When a date is sent to the database as a string (for the DATE data type) or as part of a string (for the TIMESTAMP or TIMESTAMP WITH TIME ZONE data types), the string can be specified in several different ways including that described by the ISO 8601 international standard.

A date can be specified in one of the following ISO 8601 formats.

Calendar date

The calendar date format is YYYY-MM-DD where YYYY is the year in the Gregorian calendar, MM is the month of the year between 01 (January) and 12 (December), and DD is the day of the month between 01 and 31. For example, '2010-04-01' represents the first day of April in 2010. ISO 8601 does not require the separator character. Therefore, '20100401' also represents the first day of April in 2010.

ISO calendar date	Format	Example
Basic	YYYYMMDD	20100401
Extended	YYYY-MM-DD	2010-04-01

Week date

Another ISO date format is the week date. The format is YYYY-Www-D where YYYY is the year in the Gregorian calendar, W is the letter W, ww is the week of the year between 01 (the first week) and 52 or 53 (the last week), and D is the day in the week between 1 (Monday) and 7 (Sunday). For example, '2010-W13-4' represents the fourth day of the thirteenth week of 2010 (April 1 2010). ISO 8601 does not require the separator character. Therefore, '2010W134' also represents the fourth day of the thirteenth week of 2010. For reduced accuracy, one digit can be omitted from the representation ('2010W13' represents March 29 2010).

ISO week date	Format	Example
Basic	YYYYWwwD	2010W134
Extended	YYYY-Www-D	2010-W13-4

Ordinal date

The last ISO date format is the ordinal date. The format is YYYY-DDD where YYYY is the year in the Gregorian calendar and DDD is the ordinal number of a calendar day within the calendar year. For example,

'2010-091' represents the first day of April in 2010. ISO 8601 does not require the separator character. For example, '2010091' also represents April 1 2010. The maximum ordinal date is 366 for those years with leap years. For example, '2008366' represents the last day of the year in 2008 (December 31 2008).

ISO ordinal date	Format	Example
Basic	YYYYDDD	2010091
Extended	YYYY-DDD	2010-091

Other date formats are supported. SQL Anywhere is very flexible in its interpretation of strings containing dates. Whenever there is any ambiguity, the interpretation of the date value is guided by the date_order and nearest_century database option settings. For example, depending on the date_order setting, '02/05/2002' can be interpreted by the database server as the 2nd of May (DMY), or the 5th of February (MDY), or an illegal value (YMD).

The nearest_century setting determines whether a two-digit year value is interpreted as a year in the twentieth or twenty-first century. For example, in the string '02/05/10', the date_order setting would determine whether 02 or 10 is interpreted as the year and the nearest_century setting would determine whether 02 represented 1902 or 2002, or whether 10 represented 1910 or 2010. The value of the nearest_century option affects the interpretation of 2-digit years: 2000 is added to values less than nearest_century and 1900 is added to all other values. The default value of this option is 50. So, by default, the year 50 is interpreted as 1950 and the year 49 is interpreted as 2049.

The following table shows how the first day of April in 2010 could be specified using the indicated date_order setting and a nearest_century setting of 50.

date_order	Format	Example
YMD	YYYY/MM/DD	2010/04/01
YMD	YY/MM/DD	10/04/01
MDY	MM/DD/YYYY	04/01/2010
MDY	MM/DD/YY	04/01/10
DMY	DD/MM/YYYY	01/04/2010
DMY	DD/MM/YY	01/04/10

Since ISO 8601 formats are not ambiguous and are not affected by the user's setting of date_order and nearest_century, their use is recommended.

Dates can also be specified using month names. Examples are '2010 April 01', 'April 1, 2010', and '1 April 2010'. When the year is ambiguously specified, the date_order option is used to factor the year and day of month parts. Therefore, '01 April 10' is interpreted as April 10 2001 when the date_order is 'YMD' or as April 1 2010 when the date_order is 'DMY'.

The year in a date can range from 0001 to 9999. The minimum date is 0001-01-01.

If a string contains only a partial date specification, default values are used to fill out the date. The following defaults are used:

year

The current year is used when no year is specified (for example, 'April 1').

month

The current month is used when no year and month are specified (for example, '23:59:59') or 01 if a year is specified (for example, '2010').

day

The current day is used when no year and month are specified (for example, '23:59:59') or 01 if a month is specified (for example, 'April').

In the following example, the date value is constructed from the current date.

```
SELECT CAST('23:59:59' AS TIMESTAMP);
```

Related Information

Comparisons of Dates and Times [page 197]
DATE Data Type [page 168]
date_format Option
nearest_century Option
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]

1.2.5.2.2 Time Formats

The time of day can be specified in the ISO 8601 format, using the 24-hour timekeeping system.

It is hh:mm:ss, where hh is the number of complete hours that have passed since midnight, mm is the number of complete minutes since the start of the hour, and ss is the number of complete seconds since the start of the minute. For example, '23:59:59' represents the time one second before midnight.

The ISO 8601 standard allows for the omission of seconds and minutes. For example, '23:59' represents the time sixty seconds before midnight.

The ISO 8601 standard also allows you to include a decimal fraction to the seconds unit. Fractional seconds are specified using a comma (,) or a period (.). The fraction is stored to a maximum of six decimal places. For example, '23:59:59,500000' and '23:59:59.500000' both represent the time one-half second before midnight. Fractional minutes or hours are not supported.

ISO 8601 does not require the colon separator character when the time of day is included with a date specification. For example, '235959' represents the time one second before midnight.

The maximum time of day is '24:00:00'. It represents midnight. When combined with a date, it represents midnight, or 00:00:00 of the next day. For example, '2010-04-01 24:00:00' is equivalent to '2010-04-02 00:00:00'.

ISO time	Format	Example
Basic (with date)	hhmmss.ssssss	20100401 235959.500000

ISO time	Format	Example
Basic (with date)	hhmmss,ssssss	20100401 235959,500000
Extended	hh:mm:ss.ssssss	23:59:59.500000
Extended	hh:mm:ss,ssssss	23:59:59,500000

The non-ISO AM and PM designators are also supported. For example, '11:59:59 PM' is equivalent to '23:59:59'.

AM/PM	Format	Example
AM	hh:mm:ss.ssssss AM	11:59:59.500000 AM
AM	hh:mm:ss,ssssss AM	11:59:59,500000 AM
PM	hh:mm:ss.ssssss PM	11:59:59.500000 PM
PM	hh:mm:ss,ssssss PM	11:59:59,500000 PM

Related Information

Comparisons of Dates and Times [page 197]
DATE Data Type [page 168]
date_format Option
nearest_century Option
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]

1.2.5.2.3 Date with Time Formats

ISO 8601 permits the date and time of day to be combined using a space character or the letter T.

For example, '2010-04-01 23:59:59' and '2010-04-01T23:59:59' both represent the time one second before midnight on the first day of April in 2010. The hyphen and colon separator characters can be omitted. For example, '20100401T235959' also represents the same date and time. As an extension to this format, the omission of the date and time separator are also supported. For example, '20100401235959' also represents the same date and time.

Mixing basic and extended date and time formats is supported. For example, '20100401T23:59:59' combines both the basic and extended formats.

Related Information

Comparisons of Dates and Times [page 197]
DATE Data Type [page 168]

date_format Option
nearest_century Option
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]

1.2.5.2.4 Time Zone Formats

ISO 8601 also permits the addition of a time zone offset to a date and time of day string.

The format is one of:

Z

(Zulu) The date and time of day are in Coordinated Universal Time (UTC). For example, '2010-04-01 23:00:00Z' represents 11:00 PM Coordinated Universal Time on the first day of April in 2010.

+hh:mm

The specified date and time of day are the indicated number of hours and minutes ahead of UTC. For example, '2010-04-01 23:00:00+04:00' represents 11:00 PM on the first day of April in 2010 in a time zone 4 hours east of UTC.

-hh:mm

The specified date and time of day are the indicated number of hours and minutes behind UTC. For example, '2010-04-01 23:00:00-05:00' represents 11:00 PM on the first day of April in 2010 in a time zone 5 hours west of UTC.

If the minutes are 0, it is not necessary to specify them in the time zone offset. Also, a space can precede the time zone offset. For example, '2010-04-01 23:00:00 -03:30' represents 11:00 PM on the first day of April in 2010 in a time zone three and a half hours west of UTC.

ISO time zone	Format	Example
Basic	Z	20100401 235959Z
Basic	+hhmm	20100401 235959+0400
Basic	+hh	20100401 235959+04
Basic	-hhmm	20100401 235959-0500
Basic	-hh	20100401 235959-05
Basic	using T, fraction	20100401T235959.50-0330
Extended	Z	2010-04-01 23:59:59Z
Extended	+hh:mm	2010-04-01 23:59:59+04:00
Extended	-hh:mm	2010-04-01 23:59:59-05:00
Extended	using T, fraction	2010-04-01T23:59:59.50-03:30

Mixing basic and extended date, time, and time zone formats is supported. For example, '20100401T23:59:59-05' combines both basic and extended formats.

Related Information

Comparisons of Dates and Times [page 197]
DATE Data Type [page 168]
date_format Option
nearest_century Option
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]

1.2.5.3 Retrieval of Dates and Times from the Database

Dates and times can be retrieved from the database as a string using any interface, as a string. They can also be retrieved using ODBC or OLE DB, as a binary value (using an ODBC TIMESTAMP_STRUCT structure for example), or using Embedded SQL, as a SQLDATETIME structure

Date and time of day with a time zone offset can be retrieved from the database as a string only.

When a date or time, with or without a time zone offset, is retrieved as a string, it is retrieved in the format specified by the database options date_format, time_format, timestamp_format, and timestamp_with_time_zone_format.

The following arithmetic operators are allowed on dates:

timestamp + integer

Add the specified number of days to a date or timestamp.
timestamp - integer

Subtract the specified number of days from a date or timestamp.
date - date

Compute the number of days between two dates or timestamps.
date + time

Create a timestamp combining the given date and time.

Leap Years

The database server uses a globally accepted algorithm for determining which years are leap years. Using this algorithm, a year is considered a leap year if it is divisible by four, unless the year is a century date (such as the year 1900), in which case it is a leap year only if it is divisible by 400.

All leap years are handled correctly. For example, the following SQL statement results in a return value of "Tuesday":

```
SELECT DAYNAME('2000-02-29');
```

The database server accepts February 29, 2000 (a leap year)as a date, and using this date determines the day of the week.

However, the following statement is rejected by the database server:

```
SELECT DAYNAME('2001-02-29');
```

This statement results in an error (cannot convert '2001-02-29' to a date) because February 29th does not exist in the year 2001.

Related Information

Date and Time Functions [page 213]
SET OPTION Statement [page 1387]

1.2.5.4 DATE Data Type

The DATE data type stores calendar dates, such as a year, month, and day.

Syntax

DATE

Remarks

A DATE value requires 4 bytes of storage.

The format in which DATE values are retrieved as strings by applications is controlled by the date_format option setting. For example, a DATE value representing the 19th of July, 2010 can be returned to an application as 2010/07/19, or as Jul 19, 2010 depending on the date_format option setting.

UltraLite: The format in which DATE values are retrieved as strings by applications is controlled by the date_format creation parameter. For example, a DATE value representing the 19th of July, 2010 can be returned to an application as 2010/07/19, or as Jul 19, 2010 depending on the date_format creation parameter.

Standards

ANSI/ISO SQL Standard

A feature in the standard.

Transact-SQL

Supported by Adaptive Server Enterprise.

Related Information

Date and Time Functions [page 213]
Date Formats [page 162]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
DATE Function [Date and Time] [page 315]
date_format Option
date_order Option
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
ISDATE Function [Data Type Conversion] [page 425]
nearest_century Option
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP Special Value [page 111]
TIMESTAMP WITH TIME ZONE Data Type [page 177]
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.2.5.5 DATETIME Data Type

The DATETIME data type is a Transact-SQL compatible alias for TIMESTAMP, used to store date and time of day.

Syntax

DATETIME

Remarks

A DATETIME value requires 8 bytes of storage.

The format in which DATETIME values are retrieved as strings by applications is controlled by the timestamp_format option setting. For example, the DATETIME value 2010/04/01T23:59:59.999999 can be returned to an application as 2010/04/01 23:59:59, or as April 1, 2010 23:59:59.999999 depending on the timestamp_format option setting.

> **i Note**
>
> Although the range of possible dates for the DATETIME data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the DATETIME date type is from 0001-01-01 00:00:00 up to, but not including, 7911-01-01 00:00:00. Beyond this range, the hours and minutes portion of the DATETIME value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

When a DATETIME value is converted to a DATETIMEOFFSET, the connection's time_zone_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a DATETIMEOFFSET value is converted to DATETIME, the offset is discarded.

UltraLite: When a DATETIME value is converted to DATETIMEOFFSET, the local time zone offset on the system is used in the final result.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

DATETIME, rather than TIMESTAMP, is used by Adaptive Server Enterprise. The DATETIME type in Adaptive Server Enterprise supports dates between January 1, 1753 and December 31, 9999 and supports less precision with the time portion of the value. In SQL Anywhere, DATETIME is implemented as a TIMESTAMP without these restrictions. You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise.

Related Information

Date and Time Functions [page 213]
Ways to Send Dates and Times to the Database [page 161]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Function [Date and Time] [page 315]
date_format Option
date_order Option
DATE Data Type [page 168]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]

timestamp_format Option
nearest_century Option
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.2.5.6 DATETIMEOFFSET Data Type

The DATETIMEOFFSET data type is a Transact-SQL compatible alias for TIMESTAMP WITH TIME ZONE, used to store date, time of day, and time zone information.

Syntax

DATETIMEOFFSET

Remarks

The DATETIMEOFFSET value contains the year, month, day, hour, minute, second, fraction of a second, and number of hours and minutes before or after Coordinated Universal Time (UTC). The fraction is stored to 6 decimal places.

A DATETIMEOFFSET value requires 10 bytes of storage.

The format in which DATETIMEOFFSET values are retrieved as strings by applications is controlled by the timestamp_with_time_zone_format setting . For example, the DATETIMEOFFSET value 2010/04/01T23:59:59.999999-6:00 can be returned to an application as 2010/04/01 23:59:59 -06:00 or as April 1, 2010 23:59:59.999999 -06:00, depending on the timestamp_with_time_zone_format setting.

Note

Although the range of possible dates for the DATETIMEOFFSET data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the DATETIMEOFFSET date type is from 0001-01-01 00:00:00 up to, but not including, 7911-01-01 00:00:00. Beyond this range, the hours and minutes portion of the DATETIMEOFFSET value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

Do not use DATETIMEOFFSET for computed columns or in materialized views because the value of the governing time_zone_adjustment option varies between connections based on their location and the time of year.

Two DATETIMEOFFSET values are considered identical when they represent the same instant in UTC, regardless of the TIME ZONE offset applied. For example, the following statement returns Yes because the results are considered identical:

```
SELECT
IF CAST('2009-07-15 08:00:00 -08:00' AS DATETIMEOFFSET) =
   CAST('2009-07-15 11:00:00 -05:00' AS DATETIMEOFFSET)
   THEN 'Yes'
```

```
   ELSE 'No'
END IF;
```

If you omit the time zone offset from a DATETIMEOFFSET value, it defaults to the current UTC offset of the client regardless of whether the timestamp represents a date and time in standard time or daylight time. For example, if the client is located in the Eastern Standard time zone and executes the following statement while daylight time is in effect, then a timestamp with a time zone appropriate for the Atlantic Standard time zone (-4 hours from UTC) is returned.

```
SELECT CAST('2009/01/30 12:34:55' AS DATETIMEOFFSET)
```

Comparing DATETIMEOFFSET with other data types

The comparison of DATETIMEOFFSET values with timestamps without time zones is not recommended because the default time zone offset of the client varies with the geographic location of the client and with the time of the year.

Execute the following statement to determine the current time zone offset in minutes for a client:

```
SELECT CONNECTION_PROPERTY( 'TimeZoneAdjustment' );
```

UltraLite: The TimeZoneAdjustment connection property is not supported in UltraLite databases.

Converting to or from DATETIMEOFFSET

When a DATETIME value is converted to DATETIMEOFFSET, the connection's time_zone_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a DATETIMEOFFSET value is converted to DATETIME, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

UltraLite: When a DATETIME value is converted to DATETIMEOFFSET, the client's time zone is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a DATETIMEOFFSET value is converted to DATETIME, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

Standards

ANSI/ISO SQL Standard

The specific use of DATETIMEOFFSET is not in the standard. To be compatible with the ANSI/ISO SQL Standard, use TIMESTAMP WITH TIME ZONE. The TIMESTAMP WITH TIME ZONE type is optional ANSI/ISO SQL Language Feature F411.

Related Information

Date and Time Functions [page 213]
Ways to Send Dates and Times to the Database [page 161]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]

DATE Data Type [page 168]
DATETIME Data Type [page 169]
DATE Function [Date and Time] [page 315]
DATETIME Function [Date and Time] [page 324]
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]
timestamp_with_time_zone_format Option
nearest_century Option
date_order Option
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.2.5.7 SMALLDATETIME Data Type

SMALLDATETIME is a domain, implemented as TIMESTAMP, used to store date and time of day information. SMALLDATETIME is a Transact-SQL type.

Syntax

SMALLDATETIME

Remarks

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

SMALLDATETIME is supported by Adaptive Server Enterprise. In Adaptive Server Enterprise, the SMALLDATETIME type supports dates between January 1, 1900 and June 6, 2079 and supports less

precision with the time portion of the value. In SQL Anywhere, SMALLDATETIME is implemented as a TIMESTAMP without these restrictions. You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise.

Related Information

Ways to Send Dates and Times to the Database [page 161]
Date and Time Functions [page 213]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATETIME Data Type [page 169]
DATE Function [Date and Time] [page 315]
DATETIME Function [Date and Time] [page 324]
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
TIME Data Type [page 174]
TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]
timestamp_format Option
timestamp_with_time_zone_format Option
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.2.5.8 TIME Data Type

The TIME data type stores the time of day, containing the hour, minute, second, and fraction of a second.

Syntax

TIME

Remarks

A TIME value requires 8 bytes of storage.

When using ODBC, a TIME value sent or retrieved as a binary value (using an ODBC TIME_STRUCT structure) is restricted to an accuracy of hours, minutes, and seconds. Fractional seconds are not part of the structure. For this reason, TIME values should be sent or retrieved as strings if increased accuracy is desired. The format in which TIME values are retrieved as strings by applications is controlled by the time_format option setting. For example, the TIME value 23:59:59.999999 can be returned to an application as 23:59:59, 23:59:59.999, or 23:59:59.999999 depending on the time_format option setting.

UltraLite: The format in which TIME values are retrieved as strings by applications is controlled by the time_format creation parameter. For example, the TIME value 23:59:59.999999 can be returned to an application as 23:59:59, 23:59:59.999, or 23:59:59.999999 depending on the time_format creation parameter.

Standards

ANSI/ISO SQL Standard

A feature in the standard.

Transact-SQL

The TIME data type is supported by Adaptive Server Enterprise. However, Adaptive Server Enterprise supports millisecond resolution (three digits) rather than microsecond resolution (six digits). You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise. To migrate TIME values, use the Adaptive Server Enterprise BIGTIME data type.

Related Information

Time Formats [page 164]
Date and Time Functions [page 213]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATETIME Data Type [page 169]
DATE Function [Date and Time] [page 315]
DATETIME Function [Date and Time] [page 324]
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
TIMESTAMP WITH TIME ZONE Data Type [page 177]
timestamp_format Option
timestamp_with_time_zone_format Option

UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.2.5.9 TIMESTAMP Data Type

The TIMESTAMP data type stores a point in time containing the year, month, day, hour, minute, second, and fraction of a second stored to 6 decimal places.

Syntax

TIMESTAMP

Remarks

A TIMESTAMP value requires 8 bytes of storage.

The format in which TIMESTAMP values are retrieved as strings by applications is controlled by the timestamp_format setting. For example, the TIMESTAMP value 2010/04/01T23:59:59.999999 can be returned to an application as 2010/04/01 23:59:59 or as April 1, 2010 23:59:59.999999, depending on the timestamp_format setting.

Note

Although the range of possible dates for the TIMESTAMP data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the TIMESTAMP date type is from 0001-01-01 00:00:00 up to, but not including, 7911-01-01 00:00:00. Beyond this range, the hours and minutes portion of the TIMESTAMP value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the connection's time_zone_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a TIMESTAMP WITH TIME ZONE value is converted to TIMESTAMP, the offset is discarded.

UltraLite: When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the local time zone offset on the system is used in the final result.

Standards

ANSI/ISO SQL Standard

Compatible with the standard.

Transact-SQL

Adaptive Server Enterprise uses the DATETIME type for TIMESTAMP values.

Related Information

Ways to Send Dates and Times to the Database [page 161]
Date and Time Functions [page 213]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATETIME Data Type [page 169]
DATE Function [Date and Time] [page 315]
DATETIME Function [Date and Time] [page 324]
date_order Option
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
nearest_century Option
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Special Value [page 111]
TIMESTAMP WITH TIME ZONE Data Type [page 177]
timestamp_format Option
timestamp_with_time_zone_format Option
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.2.5.10 TIMESTAMP WITH TIME ZONE Data Type

The TIMESTAMP WITH TIME ZONE data type stores a point in time with a time zone offset.

Syntax

TIMESTAMP WITH TIME ZONE

Remarks

The TIMESTAMP WITH TIME ZONE value contains the year, month, day, hour, minute, second, fraction of a second, and number of hours and minutes before or after Coordinated Universal Time (UTC). The fraction is stored to 6 decimal places.

A TIMESTAMP WITH TIME ZONE value requires 10 bytes of storage.

The format in which TIMESTAMP WITH TIME ZONE values are retrieved as strings by applications is controlled by the timestamp_with_time_zone_format setting . For example, the TIMESTAMP WITH TIME ZONE value 2010/04/01T23:59:59.999999-6:00 can be returned to an application as 2010/04/01 23:59:59 -06:00 or as April 1, 2010 23:59:59.999999 -06:00, depending on the timestamp_with_time_zone_format setting.

> **i** Note
>
> Although the range of possible dates for the TIMESTAMP WITH TIME ZONE data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the TIMESTAMP WITH TIME ZONE date type is from 0001-01-01 00:00:00 up to, but not including, 7911-01-01 00:00:00. Beyond this range, the hours and minutes portion of the TIMESTAMP WITH TIME ZONE value is not retained, but seconds and fractional seconds are. In this case, built-in functions that pertain to minutes or seconds may produce meaningless results.

Do not use TIMESTAMP WITH TIME ZONE for computed columns or in materialized views because the value of the governing time_zone_adjustment option varies between connections based on their location and the time of year.

Two TIMESTAMP WITH TIME ZONE values are considered identical when they represent the same instant in UTC, regardless of the TIME ZONE offset applied. For example, the following statement returns Yes because the results are considered identical:

```
SELECT
IF CAST('2009-07-15 08:00:00 -08:00' AS TIMESTAMP WITH TIME ZONE) =
   CAST('2009-07-15 11:00:00 -05:00' AS TIMESTAMP WITH TIME ZONE)
   THEN 'Yes'
   ELSE 'No'
END IF;
```

If you omit the time zone offset from a TIMESTAMP WITH TIME ZONE value, it defaults to the current UTC offset of the client regardless of whether the timestamp represents a date and time in standard time or daylight time. For example, if the client is located in the Eastern Standard time zone and executes the following statement while daylight time is in effect, then a timestamp with a time zone appropriate for the Atlantic Standard time zone (-4 hours from UTC) is returned.

```
SELECT CAST('2009/01/30 12:34:55' AS TIMESTAMP WITH TIME ZONE)
```

Comparing TIMESTAMP WITH TIME ZONE with other data types

The comparison of TIMESTAMP WITH TIME ZONE values with timestamps without time zones is not recommended because the default time zone offset of the client varies with the geographic location of the client and with the time of the year.

Execute the following statement to determine the current time zone offset in minutes for a client:

```
SELECT CONNECTION_PROPERTY( 'TimeZoneAdjustment' );
```

UltraLite: The TimeZoneAdjustment connection property is not supported in UltraLite databases.

Converting to or from TIMESTAMP WITH TIME ZONE

When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the connection's time_zone_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a TIMESTAMP WITH TIME ZONE value is converted to TIMESTAMP, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

UltraLite: When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the client's time zone is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a TIMESTAMP WITH TIME ZONE value is converted to TIMESTAMP, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

Standards

ANSI/ISO SQL Standard

TIMESTAMP WITH TIME ZONE is part of optional ANSI/ISO SQL Language Feature F411.

Related Information

Comparisons of Dates and Times [page 197]
Ways to Send Dates and Times to the Database [page 161]
Date and Time Functions [page 213]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATETIME Data Type [page 169]
DATE Function [Date and Time] [page 315]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
date_order Option
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
nearest_century Option
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
timestamp_format Option
timestamp_with_time_zone_format Option

UTC TIMESTAMP Special Value [page 113]

1.2.6 Binary Data Types

Binary data types store binary data, including images and other types of information that are not interpreted by the database.

In this section:

1.2.6.1 BINARY Data Type

The BINARY data type stores binary data of a specified maximum length (in bytes).

Syntax

```
BINARY [ ( max-length ) ]
```

Parameters

max-length

The maximum length of the value, in bytes. If the length is not specified, then it is 1.

The length must be in the 1 to 32767 range.

Remarks

During comparisons, BINARY values are compared exactly byte for byte. This differs from the CHAR data type, where values are compared using the collation sequence of the database.

If one binary string is a prefix of the other, the shorter string is considered to be less than the longer string.

Unlike CHAR values, BINARY values are not transformed during character set conversion.

BINARY is semantically equivalent to VARBINARY. It is a variable-length type. In other database management systems, BINARY is a fixed-length type.

UltraLite: BINARY is a domain, implemented as VARBINARY.

Standards

ANSI/ISO SQL Standard

SQL Language Feature T021.

Related Information

String Functions [page 222]
VARBINARY Data Type [page 184]
LONG BINARY Data Type [page 182]
Bitwise Operators [page 33]

1.2.6.2 IMAGE Data Type

The IMAGE data type stores binary data of arbitrary length.

Syntax

IMAGE

Remarks

IMAGE is a domain, implemented as LONG BINARY.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

String Functions [page 222]
LONG BINARY Data Type [page 182]

1.2.6.3 LONG BINARY Data Type

The LONG BINARY data type stores binary data of arbitrary length.

Syntax

LONG BINARY

Remarks

The maximum size in bytes is 2 GB minus 1 byte (2^{31} - 1) or or 2 147 483 647.

UltraLite:

- You can cast strings to/from LONG BINARY data.
- LONG BINARY data cannot be concatenated.
- LONG BINARY columns can be included in the result set of a SELECT query.
- Indexes cannot be created on a LONG BINARY type.
- A LONG BINARY type can only be used in the LENGTH and CAST functions.
- Conditions in SQL statements, such as in the WHERE clause, cannot operate on LONG BINARY columns.
- Only INSERT, UPDATE, and DELETE operations are allowed on LONG BINARY column.

Standards

ANSI/ISO SQL Standard

The LONG BINARY data type comprises SQL Language Features T021, "BINARY and VARBINARY data types", and T041, "Basic LOB data type support".

Related Information

BINARY Data Type [page 180]
VARBINARY Data Type [page 184]

1.2.6.4 UNIQUEIDENTIFIER Data Type

The UNIQUEIDENTIFIER data type stores UUID (also known as GUID) values.

Syntax

UNIQUEIDENTIFIER

Remarks

The UNIQUEIDENTIFIER data type is typically used for a primary key or other unique column to hold UUID (Universally Unique Identifier) values that uniquely identify rows. The NEWID function generates UUID values in such a way that a value produced on one computer does not match a UUID produced on another computer. UNIQUEIDENTIFIER values generated using NEWID can therefore be used as keys in a synchronization environment.

For example:

```
CREATE TABLE T1 (
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    c1 INT );
```

UUID values are also referred to as GUID (Globally Unique Identifier) values.

UNIQUEIDENTIFIER values are stored as BINARY(16) but are described to client applications as BINARY(36). This description ensures that if the client fetches the value as a string, it has allocated enough space for the result.

For SQL Anywhere ODBC client applications, uniqueidentifier values appear as a SQL_GUID type.

UNIQUEIDENTIFIER values are automatically converted between string and binary values as needed. Input string values may contain hyphens, but must be properly formatted if they do. The following illustrates two permissible input formats.

```
SELECT STRTOUUID('9752b904beef4bd8adb5642ea2c71986'),
       STRTOUUID('9752b904-beef-4bd8-adb5-642ea2c71986');
```

UNIQUEIDENTIFIER string values are formatted with hyphens so they are compatible with other RDBMSs.

You can change this by setting the uuid_has_hyphens option to Off.

UltraLite: There is no comparable setting for UltraLite. UNIQUEIDENTIFIER string values are always formatted with hyphens. Input string values must be properly formatted with hyphens.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

The NEWID Default
String Functions [page 222]
NEWID Function [Miscellaneous] [page 467]
UUIDTOSTR Function [String] [page 615]
STRTOUUID Function [String] [page 575]
uuid_has_hyphens Option

1.2.6.5 VARBINARY Data Type

The VARBINARY data type stores binary data of a specified maximum length (in bytes).

Syntax

```
VARBINARY [ ( max-length ) ]
```

Parameters

max-length

The maximum length of the value, in bytes. If the length is not specified, then it is 1.

The length must be in the 1 to 32767 range.

Remarks

During comparisons, VARBINARY values are compared exactly byte for byte. This behavior differs from the CHAR data type, where values are compared using the collation sequence of the database.

VARBINARY values are not transformed during character set conversion.

VARBINARY can also be specified as BINARY VARYING. Regardless of which syntax is used, the data type is described as VARBINARY. If one binary string is a prefix of the other, the shorter string is considered to be less than the longer string.

UltraLite: If one binary string is a prefix of the other, the shorter string is compared to the other as though the shorter string were padded with zeros. When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.

Standards

ANSI/ISO SQL Standard

SQL Language Feature T021, "BINARY and VARBINARY data types".

Related Information

String Functions [page 222]
BINARY Data Type [page 180]
LONG BINARY Data Type [page 182]
Bitwise Operators [page 33]

1.2.7 ROW and ARRAY Composite Data Types

Composite data types are values that are comprised of zero or more elements, where each element has a value of a particular data type. Currently only ROW and ARRAY composite data types are supported.

Composite data types are not a specific data type, they are constructors containing the definition for how to assemble one or more data types into a row. When you define a ROW or ARRAY type, you are defining the number and data type of each element that make up a row (an array is a set of these rows).

ROWS and ARRAYS are a more efficient way to store lists because they define the structure and data type of their values. They also simplify the creation of list elements, either directly, by using double square brackets; or as result set, by using the UNNEST operator. Consider using the ARRAY data type if you are storing lists as delimited strings in VARCHAR columns, and parsing them using sa_split_list. ARRAYS are helpful when storing different objects that are all related. ROWS are helpful when storing multiple values related to one object.

Creating Domains of ROW or ARRAY Type

You can create domains of ROW or ARRAY type. For example, the following statement creates a domain called MyRow, and defines its composite type as two integer values.

```
CREATE DOMAIN MyRow ROW( a INT, b INT );
```

Creating a ROW or ARRAY domain allows you to reference the row or array type, instead of defining it repeatedly inside statements.

Declaring an ARRAY Type

An ARRAY type is a homogeneous, ordered collection that can be passed in whole or in part as an argument to SQL stored procedures or functions.

An ARRAY type can consist of up to 6.4 million elements. An ARRAY is initialized to a zero-length ARRAY of the declared type.

An ARRAY constructor builds an ARRAY value so the array can be processed in a query or passed as an argument to a SQL stored procedure or function.

The database server supports the following syntaxes for declaring variables of the ARRAY type:

```
DECLARE variable-name element-type-name ARRAY [ ( maximum-size ) ]
```

```
DECLARE variable-name ARRAY [ ( maximum-size ) ] OF element-type-name
```

If `maximum-size` is omitted, then the array can contain up to 6.4 million elements. Variables of the ARRAY type are not initialized to NULL the way all other non-composite variables are. Instead, they are initialized to a zero-length array.

ARRAY types cannot be stored as columns in a base or temporary table and are not supported in:

- the outermost SELECT list of a view definition
- a top-level SELECT block or query expression that is returned to the client
- a base table
- a temporary table
- an Embedded SQL FETCH statement

> **i Note**
>
> If a procedure returns a column of type ROW or ARRAY in its result set, the procedure must have a RESULT column, or an error is returned. If necessary, change the procedure to use Watcom SQL syntax.

The following example illustrates how to declare an array of 5 integers:

```
DECLARE NewArray INTEGER ARRAY( 5 );
```

The following example illustrates an alternative way of declaring an array of 5 integers that is compatible with the Oracle syntax:

```
DECLARE NewArray ARRAY( 5 ) OF INTEGER;
```

The following example illustrates how to declare a two-dimensional array, where New2DArray contains 10 elements, each of which is a five-element array of integers:

```
DECLARE New2DArray INTEGER ARRAY( 5 ) ARRAY( 10 );
```

The following example illustrates an alternative way of declaring a two-dimensional array that is compatible with the Oracle syntax, where New2DArray contains 10 elements, each of which is a five-element array of integers:

```
DECLARE New2DArray ARRAY( 10 ) OF ARRAY( 5 ) OF INTEGER;
```

Declaring a ROW Type

A ROW type is described by a row type descriptor, which consists of the field descriptor of each field in the ROW type. ROW types are restricted to 45000 fields, which is the same limit as the number of columns in a table.

Variables in the ROW type are initialized to a ROW of the declared type.

A ROW supports the construction of structured types, consisting of a group of fields of potentially different types. ROW types can be part of higher-order row types, which permits complex structures involving other row types and arrays.

The following example illustrates how to declare a variable, student, that is defined as a structured type of four different fields:

```
DECLARE student ROW( studentID INTEGER,
   student_first_name VARCHAR( 40 ),
   student_last_name VARCHAR( 50 ),
   student address LONG VARCHAR );
```

The following example illustrates how to assign a ROW as a complete structure:

```
DECLARE employee ROW( empID INTEGER,
   address ROW( street_address LONG VARCHAR,
   city VARCHAR( 50 ),
   province VARCHAR( 30 ),
   country VARCHAR( 40 )
   )
);
DECLARE temp_address ROW( street_address LONG VARCHAR,
   city VARCHAR( 50 ),
   province VARCHAR( 30 ),
   country VARCHAR( 40 )
);
SET temp_address = employee.address;
```

The following is an array of ROW example:

```
CREATE OR REPLACE VARIABLE tvar ARRAY OF ROW(
   a VARCHAR(32),
   b ARRAY OF ROW( b1 LONG NVARCHAR, b2 LONG NVARCHAR),
   c BIT,
   d NUMERIC(5,2)
   );
SET tvar[[1]] = ROW( 'rowA',
ARRAY(ROW('rowA11','rowB11'),ROW('rowA12','rowB12')), 1, 1.01);
SET tvar[[2]] = ROW( 'rowB',
ARRAY(ROW('rowA21','rowB21'),ROW('rowA22','rowB22')), 0, 1.02);
SET tvar[[3]] = ROW( 'rowC',
ARRAY(ROW('rowA31','rowB31'),ROW('rowA32','rowB32')), 1, 1.03);
SELECT tvar[[x.row_num]].a AS a,
   tvar[[x.row_num]].b[[y.row_num]].b1 AS b1,
   tvar[[x.row_num]].b[[y.row_num]].b2 AS b2,
   tvar[[x.row_num]].c AS c,
   tvar[[x.row_num]].d AS d
   FROM sa_rowgenerator(1,CARDINALITY(tvar)) AS x,
sa_rowgenerator(1,CARDINALITY(tvar[[1]].b)) AS y;
```

Related Information

ARRAY Constructor [Composite] [page 243]
ROW Constructor [Composite] [page 534]
UNNEST Array Operator [page 28]

1.2.8 TABLE REF Data Type

The TABLE REF data type stores a reference to a base table, temporary table, or view. This data type is only for use with connection-scope variables.

Syntax

Declaring a variable of type TABLE REF

```
DECLARE table-ref-variable TABLE REF
  [ { DEFAULT | = } TABLE REF ( [ owner.]table-name ) ]
```

Creating a variable of type TABLE REF

```
CREATE VARIABLE table-ref-variable TABLE REF
  [ { DEFAULT | = } TABLE REF ( [ owner.]table-name ) ]
```

Setting a variable of type TABLE REF

```
SET table-ref-variable = TABLE REF ( table-name )
```

Referencing a variable of type TABLE REF in DML statements

```
TABLE REF ( table-ref-variable ) AS correlation-name
```

Referencing a variable of type TABLE REF as a parameter in a function or procedure

```
TABLE REF ( table-ref-variable )
```

Parameters

table-ref-variable A valid identifier for the table reference variable.
table-name The name of a base table. Optionally, include the owner as part of the specification (for example, GROUPO.Employees); this is recommended for base tables and views.

Remarks

Table reference variables (variables of type TABLE REF) allow procedures and functions to be defined even though the names of the tables they operate on change or have not yet been defined.

When referencing a variable of TABLE REF type in a DML statement, you must specify a correlation name for results.

When you specify a table reference variable in a statement, the table is looked up immediately before the statement is executed.

Table reference variables can only be accessed by their creator.

Creating a table reference variable does not create a dependence between the variable and the underlying table, and DDL statements can still be performed on tables referenced by a table reference variable.

If a table is dropped, then any table reference variables that refer to it are invalidated; an attempt to use an invalid table reference variable returns an error.

When executing a statement that acts on a table specified by using a table reference variable, you need the appropriate privileges on the underlying table referenced by the variable.

Restrictions on the use of table reference variables:

- Table reference variables cannot be used in a SELECT or DML statement if the variable resolves to the NULL value.
- Table reference variables cannot be used to specify tables in DDL statements.
- Table reference variables cannot be used as columns in base tables, temporary tables, or views.
- Table reference variables cannot be used in a top-level SELECT block or query expression that is returned to a client.
- Table reference variables cannot be combined with other types of variables in built-in functions that require a common super-type for the parameters.
- Table reference variables cannot be ordered or used as part of calculations or comparisons except for equality and inequality.

The table reference variable functionality overlaps with indirect identifier functionality; both are ways of indirectly referring to a table. However, a table reference is resolved at creation time and remains a valid reference, whereas an indirect reference is resolved when the statement that references it is executed and therefore may not be a valid reference.

Additionally, a table reference can provide access to a table that is not accessible in the current context, whereas an indirect identifier cannot. For example, suppose your procedure creates, and then refers to, a local table that has the same name as a base table. Now let's say you need to refer to the base table from within the procedure. An indirect identifier for the table would resolve to the local table, which is not what you want. To precisely identify the base table instead, use a table reference variable. For example:

```
CREATE OR REPLACE PROCEDURE PROC1()
BEGIN
    DECLARE LOCAL TEMPORARY TABLE myTab (x INTEGER, y INTEGER);
    DECLARE tab_ref TABLE REF = TABLE REF (myTab);
    INSERT INTO myTab VALUES (1,100), (2,200), (3,300);
    SELECT * FROM PROC2( tab_ref );
END;
CREATE OR REPLACE PROCEDURE PROC2( IN @tab_ref TABLE REF )
RESULT (v1 LONG VARCHAR, v2 INTEGER)
BEGIN
    DECLARE LOCAL TEMPORARY TABLE myTab ( pk INTEGER, val LONG VARCHAR );
    INSERT INTO myTab VALUES( 1, 'apple'), (3, 'pear'), (10, 'banana');
    SELECT val,T2.y FROM myTab T1 JOIN TABLE REF( @tab_ref ) AS T2 ON T2.x =
T1.pk;
END;
CALL PROC1;
```

Table 2: Result:

v1	v2
apple	100
pear	300

The myTab table in PROC2 shadows (hides) myTab that was created in PROC1, so the only table accessible by using the name myTab in PROC2 would be the locally declared myTab. Using a table reference (`TABLE REF( @tab_ref )`) more precisely identifies the object being joined to (in this example, the table created in PROC1.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example declares a table reference variable, @ref, sets it to the GROUPO.Employees table reference, and then queries the table using the table reference variable:

```
DECLARE @ref TABLE REF = TABLE REF ( GROUPO.Employees )
SELECT * FROM TABLE REF ( @ref ) AS T;
```

The following example creates a table reference variable called @tableDefinition and sets it to the GROUPO.Employees table reference, and then selects from the table using the table reference variable:

```
CREATE VARIABLE @tableDefinition TABLE REF;
SET @tableDefinition = TABLE REF ( GROUPO.Employees );
SELECT * FROM TABLE REF ( @tableDefinition ) AS T;
```

The following code snippet declares a variable named @myTableRefVariable1 with the TABLE REF data type and sets it to the GROUPO.Employees table reference:

```
DECLARE @myTableRefVariable1 TABLE REF;
SET @myTableRefVariable1 = TABLE REF ( GROUPO.Employees );
```

The following example creates a TABLE REF variable, sets it to the GROUPO.Employees table, and then queries the table reference variable for employees with birthdays in the month of February:

```
CREATE VARIABLE @myTableRefVariable2 TABLE REF;
SET @myTableRefVariable2 = TABLE REF ( GROUPO.Employees );
SELECT T.surname, T.givenname, T.birthdate FROM TABLE REF
( @myTableRefVariable2 ) AS T
    WHERE MONTH( T.birthdate ) = 2;
```

Table 3: Results

surname	givenname	birthdate
Davidson	Jo Ann	1957-02-17
Samuels	Peter	1968-02-28
Barker	Joseph	1969-02-14
Sterling	Paul	1950-02-27

The following example shows a table reference variable (@myTableRefVariable3) being used in several statements to update the GROUPO.Employees table. Notice that a correlation name (T, in this example) is required when specifying a table using a table reference variable in a DML statement:

```
CREATE VARIABLE @myTableRefVariable3 TABLE REF;
SET @myTableRefVariable3 = TABLE REF ( GROUPO.Employees );
UPDATE TABLE REF ( @myTableRefVariable3 ) AS T SET T.GivenName =
REPLACE( GivenName, 'Fran', 'Francis' );
DELETE FROM TABLE REF ( @myTableRefVariable3 ) AS T WHERE Surname = 'Holmes';
SELECT * FROM TABLE REF ( @myTableRefVariable3 ) AS T;
```

The following example shows how you can use table reference variables in a procedure:

```
CREATE OR REPLACE PROCEDURE leapday_births( IN @tab TABLE REF )
RESULT ( Surname person_name_t, GivenName person_name_t, Birthdate TIMESTAMP)
BEGIN
    SELECT Surname, GivenName, Birthdate FROM TABLE REF ( @tab ) AS T
        WHERE MONTH( Birthdate ) = 02 AND DAY( Birthdate ) = 29;
END;
CREATE VARIABLE @myTableRefVariable4 TABLE REF;
SET @myTableRefVariable4 = TABLE REF ( GROUPO.Employees );
CALL leapday_births(@myTableRefVariable4);
```

Related Information

SQL Variables [page 123]
Identifiers [page 6]
Indirect Identifiers [page 8]

1.2.9 Spatial Data Types

Many spatial data types are supported. The documentation for these data types are located with the spatial SQL API documentation.

Related Information

Supported Spatial Data Types and Their Hierarchy

1.2.10 Domains

Domains are aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are predefined, but you can add more of your own.

Domains, also called **user-defined data types**, allow columns throughout a database to be automatically defined on the same data type, with the same NULL or NOT NULL condition, with the same DEFAULT setting, and with the same CHECK condition. Domains encourage consistency throughout the database and can eliminate some types of errors.

Simple Domains

Domains are created using the CREATE DOMAIN statement.

The following statement creates a data type named street_address, which is a 35-character string.

```
CREATE DOMAIN street_address CHAR( 35 );
```

CREATE DATATYPE can be used as an alternative to CREATE DOMAIN, but is not recommended.

You must have the CREATE DATATYPE or CREATE ANY OBJECT system privilege to create domains. Once a data type is created, the user ID that executed the CREATE DOMAIN statement is the owner of that data type. Any user can use the data type. Unlike with other database objects, the owner name is never used to prefix the data type name.

The street_address data type can be used in exactly the same way as any other data type when defining columns. For example, the following table with two columns has the second column as a street_address column:

```
CREATE TABLE twocol (
    ID INT,
    street street_address
);
```

You can also drop a domain:

```
DROP DOMAIN street_address;
```

This statement can be executed only if the data type is not used in any table in the database. If you attempt to drop a domain that is in use, an error message appears.

Constraints and Defaults With Domains

Many of the attributes associated with columns, such as allowing NULL values, having a DEFAULT value, and so on, can be built into a domain. Any column that is defined on the data type automatically inherits the NULL setting, CHECK condition, and DEFAULT values. This allows uniformity to be built into columns with a similar meaning throughout a database.

For example, many primary key columns in the SQL Anywhere sample database are integer columns holding ID numbers. The following statement creates a data type that can be useful for such columns:

```
CREATE DOMAIN ID INT
NOT NULL
DEFAULT AUTOINCREMENT
CHECK( @col > 0 );
```

By default, a column created using the id data type does not allow NULLs, defaults to an auto-incremented value, and must hold a positive number. Any identifier could be used instead of `col` in the `@col` variable.

The attributes of a data type can be overridden by explicitly providing attributes for the column. A column created using the id data type with NULL values explicitly allowed does allow NULLs, regardless of the setting in the id data type.

Compatibility

Named constraints and defaults

Domains are created with a base data type, and optionally a NULL or NOT NULL condition, a default value, and a CHECK condition. Named constraints and named defaults are not supported.

Creating Data Types

You can use the sp_addtype system procedure to add a domain, or you can use the CREATE DOMAIN statement.

Related Information

CREATE DOMAIN Statement [page 836]
DROP DOMAIN Statement [page 1093]

1.2.11 Data Type Comparisons

When a comparison (such as =) is performed between arguments with different data types, one or more arguments must be converted so that the comparison operation is done using one data type.

Some rules may lead to conversions that fail, or lead to unexpected results from the comparison. In these cases, you should explicitly convert one of the arguments using CAST or CONVERT.

You can override these conversion rules by explicitly casting arguments to another type. For example, to compare a DATE and a CHAR as a CHAR, explicitly cast the DATE to a CHAR.

In this section:

When a character cannot be represented in the character set into which it is being converted, a substitution character is used instead. Conversions of this type are considered **lossy**; the original character is lost if it cannot be represented in the destination character set.

Comparisons Between CHAR and NCHAR [page 195]
When a comparison is performed between a value of CHAR type (CHAR, VARCHAR, LONG VARCHAR) and a value of NCHAR type (NCHAR, NVARCHAR, LONG NVARCHAR), the database server uses inference rules to determine the type in which the comparison should be performed.

Comparisons Between Numeric Data Types [page 196]
The database server uses the rules when comparing numeric data types. The rules are examined in the order listed, and the first rule that applies is used.

Comparisons of Dates and Times [page 197]
The table below summarizes the conversions that are implicit when comparing certain data types with date, time, or date-time data types.

Comparisons of Composite Types [page 198]
Array elements are compared starting from the first element.

Transact-SQL String to Date/Time Conversions [page 199]
If a string containing only a time value (no date) is converted to a date/time data type, the database server uses the current date.

Other Comparisons [page 199]
There are other data type comparisons that take place.

Related Information

CAST Function [Data Type Conversion] [page 275]

1.2.11.1 Lossy Conversion and Substitution Characters

When a character cannot be represented in the character set into which it is being converted, a substitution character is used instead. Conversions of this type are considered **lossy**; the original character is lost if it cannot be represented in the destination character set.

Also, not only may different character sets have a different substitution character, but the substitution character for one character set can be a non-substitution character in another character set. This is important to understand when multiple conversions are performed on a character because the final character may not appear as the expected substitution character of the destination character set.

For example, suppose that the client character set is Windows-1252, and the database character set is ISO_8859-1:1987, the U.S. default for some versions of UNIX and Linux. Then, suppose a non-Unicode client application (for example, Embedded SQL) attempts to insert the euro symbol into a CHAR, VARCHAR, or LONG VARCHAR column. Since the character does not exist in the CHAR character set, the substitution character for ISO_8859-1:1987, 0x1A, is inserted.

Now, if this same ISO_8859-1:1987 substitution character is then fetched as Unicode (for example, by doing a `SELECT * FROM t` into a SQL_C_WCHAR bound column in ODBC), this character becomes the Unicode code

point U+001A. (In Unicode the code point U+001A is the record separator control character.) However, the substitution character for Unicode is the code point U+FFFD. This example illustrates that even if your data contains substitution characters, those characters, due to multiple conversions, may not be converted to the substitution character of the destination character set.

Therefore, it is important to understand and test how substitution characters are used when converting between multiple character sets.

The on_charset_conversion_failure option can help determine the behavior during conversion when a character cannot be represented in the destination character set.

Related Information

Data Type Conversions [page 200]
Comparisons Between CHAR and NCHAR [page 195]
on_charset_conversion_failure Option

1.2.11.2 Comparisons Between CHAR and NCHAR

When a comparison is performed between a value of CHAR type (CHAR, VARCHAR, LONG VARCHAR) and a value of NCHAR type (NCHAR, NVARCHAR, LONG NVARCHAR), the database server uses inference rules to determine the type in which the comparison should be performed.

Generally, if one value is based on a column reference and the other is not, the comparison is performed in the type of the value containing the column reference.

The inference rules revolve around whether a value is based on a column reference. In the case where one value is a variable, a host variable, a literal constant, or a complex expression not based on a column reference and the other value is based on a column reference, then the constant-based value is implicitly cast to the type of the column-based value.

Following are the inference rules, in the order in which they are applied:

- If the NCHAR value is based on a column reference, the CHAR value is implicitly cast to NCHAR, and the comparison is done as NCHAR. This includes the case where both the NCHAR and CHAR value are based on column references.
- Else if the NCHAR value is not based on a column reference, and the CHAR value is based on a column reference, the NCHAR value is implicitly cast to CHAR, and the comparison is done as CHAR.
  It is important to consider the setting for the on_charset_conversion_failure option if you anticipate NCHAR to CHAR conversions since this option controls behavior if an NCHAR character cannot be represented in the CHAR character set.
- Else if neither value is based on a column reference, then the CHAR value is implicitly cast to NCHAR and the comparison is done as NCHAR.

Example

The condition `Employees.GivenName = N'Susan'` compares a CHAR column (Employees.GivenName) to the literal N'Susan'. The value N'Susan' is cast to CHAR, and the comparison is performed as if it had been written as:

```
Employees.GivenName  = CAST( N'Susan' AS CHAR );
```

Alternatively, the condition `Employees.GivenName = T.nchar_column` would find that the value T.nchar_column cannot be cast to CHAR. The comparison would be performed as if it were written as follows, and an index on Employees.GivenName cannot be used:

```
CAST( Employees.GivenName AS NCHAR ) = T.nchar_column;
```

Related Information

NCHAR to CHAR Conversions [page 201]
NCHAR to CHAR Conversions [page 201]
Lossy Conversion and Substitution Characters [page 194]
CAST Function [Data Type Conversion] [page 275]
CONVERT Function [Data Type Conversion] [page 294]
CAST Function [Data Type Conversion] [page 275]
on_charset_conversion_failure Option

1.2.11.3 Comparisons Between Numeric Data Types

The database server uses the rules when comparing numeric data types. The rules are examined in the order listed, and the first rule that applies is used.

1. If one argument is TINYINT and the other is INTEGER, convert both to INTEGER and compare.
2. If one argument is TINYINT and the other is SMALLINT, convert both to SMALLINT and compare.
3. If one argument is UNSIGNED SMALLINT and the other is INTEGER, convert both to INTEGER and compare.
4. If the data types of the arguments have a common super type, convert to the common super type and compare. The super types are the final data type in each of the following lists:
   - BIT > TINYINT > UNSIGNED SMALLINT > UNSIGNED INTEGER > UNSIGNED BIGINT > NUMERIC
   - SMALLINT > INTEGER > BIGINT > NUMERIC
   - REAL > DOUBLE
   - CHAR > LONG VARCHAR
   - BINARY > LONG BINARY

   For example, if the two arguments are of types BIT and TINYINT, they are converted to NUMERIC.

1.2.11.4 Comparisons of Dates and Times

The table below summarizes the conversions that are implicit when comparing certain data types with date, time, or date-time data types.

Data type	Data type	Conversion
CHAR	DATE	CHAR cast to TIMESTAMP; DATE cast to TIMESTAMP
CHAR	TIME	CHAR cast to TIME
CHAR	TIMESTAMP	CHAR cast to TIMESTAMP
CHAR	TIMESTAMP WITH TIME ZONE	CHAR cast to TIMESTAMP WITH TIME ZONE
DATE	TIME	illegal
DATE	TIMESTAMP	DATE cast to TIMESTAMP
DATE	TIMESTAMP WITH TIME ZONE	DATE cast to TIMESTAMP WITH TIME ZONE
DATE	SMALLINT, INTEGER, BIGINT, and NUMERIC	SMALLINT, INTEGER, BIGINT, and NUMERIC value treated as a date string and cast to TIMESTAMP; DATE cast to TIMESTAMP
DATE	REAL, FLOAT, and DOUBLE	REAL, FLOAT, and DOUBLE treated as a number of days since 0000-02-29 and cast to TIMESTAMP; DATE cast to TIMESTAMP
TIME	TIMESTAMP	TIMESTAMP cast to TIME
TIME	TIMESTAMP WITH TIME ZONE	illegal
TIMESTAMP	TIMESTAMP WITH TIME ZONE	TIMESTAMP cast to TIMESTAMP WITH TIME ZONE
TIMESTAMP	SMALLINT, INTEGER, BIGINT, and NUMERIC	SMALLINT, INTEGER, BIGINT, and NUMERIC value treated as a date string and cast to TIMESTAMP
TIMESTAMP	REAL, FLOAT, and DOUBLE	REAL, FLOAT, and DOUBLE treated as a number of days since 0000-02-29 and cast to TIMESTAMP

The following points expand on the information presented in the table above.

1. Only values of type TIME, TIMESTAMP, and CHAR can be compared to a value of type TIME. Comparison with values of other data types results in a conversion error. When comparing a time value and a value of another type, the comparison data type is TIME.
2. When comparing a TIMESTAMP, SMALLINT, INTEGER, BIGINT, NUMERIC, REAL, FLOAT, or DOUBLE value to a DATE value, the comparison data type is always TIMESTAMP.
3. When comparing a TIMESTAMP WITH TIME ZONE value to a DATE value, the comparison data type is TIMESTAMP WITH TIME ZONE.
4. When a time value is cast to a TIMESTAMP, the result is formed by combining the current date with the time value.

5. Exact numeric values of type SMALLINT, INTEGER, BIGINT, and NUMERIC can be converted to date values. The conversion is performed by treating the number as a string. For example, the integer value 20100401 represents the first day of April in 2010.
6. The unsigned exact numeric types BIT, TINYINT, UNSIGNED SMALLINT, UNSIGNED INTEGER, and UNSIGNED BIGINT cannot be converted to date values.
7. Approximate numeric values of type REAL, FLOAT, and DOUBLE can be converted to dates by treating the number as the number of days since the fictional date 0000-02-29. For example, 307 represents 0001-01-01 and 734169 represents 2010-04-01.

Related Information

Date and Time Data Types [page 160]

1.2.11.5 Comparisons of Composite Types

Array elements are compared starting from the first element.

When a difference is found, the comparison stops and the result of the comparison between the most recently compared elements is returned. If all of the elements compare equal, then the arrays are equal. The comparisons performed are equivalent to those performed on expressions that are not held in arrays. If one array is shorter than another, and all elements of the shorter array are equal to the same elements of the longer array, the shorter array is considered less than the longer array.

When comparing arrays, the arrays must hold values with union-compatible data types. Duplicate elimination and GROUP BY are also supported over array expressions. For example, with the following array comparison, the query returns 1:

```
SELECT IF ARRAY(3,4,5) > ARRAY(2,3,4) THEN 1 ELSE 0 ENDIF;
```

Row types can be compared, used in joins, duplicate elimination, and grouping. Consider two row types similar to the row expression sample above:

```
BEGIN
DECLARE test1 ROW(x INT, w ROW(y INT, z INT));
DECLARE test2 ROW(a INT, b ROW(c INT, d CHAR(3)));
SET test1 = ROW(3, ROW(6,7));
SET test2 = ROW(3, ROW(8,'7'));
SELECT (IF (test1 > test2) THEN 1 ELSE 0 ENDIF) AS RESULT FROM SYS.DUMMY;
END
```

Two row expressions can be compared only if their structures match. However, while the row expressions must have the same structure, the names of the attributes of a row type do not need to be identical, and the data types of the individual leaf values do not need to be identical, only union compatible.

All ROW comparisons other than equality and inequality operations result in UNKNOWN.

1.2.11.6 Transact-SQL String to Date/Time Conversions

If a string containing only a time value (no date) is converted to a date/time data type, the database server uses the current date.

If the fraction portion of a time is less than 3 digits, the database server interprets the value the same way regardless of the whether it is preceded by a period or a colon: one digit means tenths, two digits mean hundredths, and three digits mean thousandths.

Example

The database server converts the milliseconds value in the same manner regardless of the separator.

```
12:34:56.7 to 12:34:56.700
12:34:56:7 to 12:34:56.700
12.34.56.78 to 12:34:56.780
12.34.56:78 to 12:34:56.780
12:34:56.789 to 12:34:56.789
12:34:56:789 to 12:34:56.789
```

1.2.11.7 Other Comparisons

There are other data type comparisons that take place.

1. If the data types are a mixture of CHAR (such as CHAR, VARCHAR, LONG VARCHAR, and so on, but not NCHAR types), convert to LONG VARCHAR and compare.
2. If the data type of any argument is UNIQUEIDENTIFIER, convert to UNIQUEIDENTIFIER and compare.
3. If the data type of any argument is a bit array (VARBIT or LONG VARBIT), convert to LONG VARBIT and compare.
4. If one argument has CHARACTER data type and the other has BINARY data type, convert to BINARY and compare.
5. If one argument is a CHAR type, and the other argument is an NCHAR type, use predefined inference rules.
6. If no rule exists, convert to NUMERIC and compare.
   For example, if the two arguments have REAL and CHAR data types, they are both converted to NUMERIC.

Because of the characteristics of the values being stored, some data types such as ARRAY and TABLE REF may not return expected results when used in comparisons, even when being compared against values of the same type. In the case of the TABLE REF data type, which is only supported for variables, two values of type TABLE REF are considered equal if they contain an identical value and have the same owner.

Related Information

Comparisons Between CHAR and NCHAR [page 195]

1.2.12 Data Type Conversions

Type conversions can happen automatically, or they can be explicitly requested using the CAST or CONVERT function. The following functions can also be used to force type conversions:

DATE function

Converts the expression into a DATE, and removes any hours, minutes or seconds. Conversion errors can be reported.

DATETIME function

Converts the expression into a TIMESTAMP, and removes any time zone. Conversion errors can be reported.

STRING function

This function is equivalent to CAST(value AS LONG VARCHAR).

VALUE+0.0

Equivalent to CAST(value AS DECIMAL).

The following list is a high-level view of automatic data type conversions:

- If a string is used in a numeric expression or as an argument to a function that expects a numeric argument, the string is converted to a number.
- If a number is used in a string expression or as a string function argument, it is converted to a string before being used.
- All date constants are specified as strings. The string is automatically converted to a date before use.

Some data types, such as the TABLE REF data type (for use only with variables), cannot be converted to or from another data type.

There are certain cases where the automatic database conversions are not appropriate. For example, the automatic data type conversion fails in the example below.

```
'12/31/90' + 5
'a' > 0
```

In this section:

NCHAR to CHAR Conversions [page 201]
NCHAR to CHAR conversions can occur as part of a comparison of CHAR and NCHAR data, or when specifically requested.

NULL Constant Conversions to NUMERIC and String Types [page 202]
When converting a NULL constant to a NUMERIC, or to a string type such as CHAR, VARCHAR, LONG VARCHAR, BINARY, VARBINARY, and LONG BINARY the size is set to 0. For example:

Bit Array Conversions [page 202]
You can convert to and from bit arrays.

Numeric Set Conversions [page 204]
When converting a DOUBLE type to a NUMERIC type, precision is maintained for the first 15 significant digits.

Java and SQL Data Type Conversions [page 204]

Data type conversion between Java types and SQL types is required for both Java stored procedures and JDBC applications. Java to SQL and SQL to Java data type conversions are carried out according to the JDBC standard. The conversions are described in the following tables.

Related Information

Data Type Conversion Functions [page 212]
DATE Function [Date and Time] [page 315]
DATETIME Function [Date and Time] [page 324]
STRING Function [String] [page 574]
CAST Function [Data Type Conversion] [page 275]

1.2.12.1 NCHAR to CHAR Conversions

NCHAR to CHAR conversions can occur as part of a comparison of CHAR and NCHAR data, or when specifically requested.

This type of conversion is lossy because depending on the CHAR character set, there can be some NCHAR characters that cannot be represented in the CHAR type. When an NCHAR character cannot be converted to CHAR, a substitution character from the CHAR character set is used instead. For single-byte character sets, this is usually hex 1A.

Depending on the setting of the on_charset_conversion_failure option, when a character cannot be converted, one of the following can happen:

- a substitute character is used, and no warning is issued
- a substitute character is used, and a warning is issued
- an error is returned

Therefore, it is important to consider this option when converting from NCHAR to CHAR.

Related Information

Comparisons Between CHAR and NCHAR [page 195]
on_charset_conversion_failure Option

1.2.12.2 NULL Constant Conversions to NUMERIC and String Types

When converting a NULL constant to a NUMERIC, or to a string type such as CHAR, VARCHAR, LONG VARCHAR, BINARY, VARBINARY, and LONG BINARY the size is set to 0. For example:

`SELECT CAST( NULL AS CHAR )` returns CHAR(0)

`SELECT CAST( NULL AS NUMERIC )` returns NUMERIC(1,0)

1.2.12.3 Bit Array Conversions

You can convert to and from bit arrays.

Converting Integers to Bit Arrays

When converting an integer to a bit array, the length of the bit array is the number of bits in the integer type, and the bit array's value is the binary representation. The most significant bit of the integer becomes the first bit of the array.

Example

`SELECT CAST( CAST( 1 AS BIT ) AS VARBIT )` returns a VARBIT(1) containing 1.

`SELECT CAST( CAST( 8 AS TINYINT ) AS VARBIT )` returns a VARBIT(8) containing 00001000.

`SELECT CAST( CAST( 194 AS INTEGER ) AS VARBIT )` returns a VARBIT(32) containing 00000000000000000000000011000010.

Converting Binary to Bit Arrays

When converting a binary type of length `n` to a bit array, the length of the array is `n` * 8 bits. The first 8 bits of the bit array become the first byte of the binary value. The most significant bit of the binary value becomes the first bit in the array. The next 8 bits of the bit array become the second byte of the binary value, and so on.

Example

`SELECT CAST( 0x8181 AS VARBIT )` returns a VARBIT(16) containing 1000000110000001.

Converting Characters to Bit Arrays

When converting a character data type of length `n` to a bit array, the length of the array is `n` bits. Each character must be either '0' or '1' and the corresponding bit of the array is assigned the value 0 or 1.

Example

`SELECT CAST( '001100' AS VARBIT )` returns a VARBIT(6) containing 001100.

Converting Bit Arrays to Integers

When converting a bit array to an integer data type, the bit array's binary value is interpreted according to the storage format of the integer type, using the most significant bit first.

Example

`SELECT CAST( CAST( '11000010' AS VARBIT ) AS INTEGER )` returns 194 (11000010_2 = 0xC2 = 194).

Converting Bit Arrays to Binary

When converting a bit array to a binary, the first 8 bits of the array become the first byte of the binary value. The first bit of the array becomes the most significant bit of the binary value. The next 8 bits are used as the second byte, and so on. If the length of the bit array is not a multiple of 8, then extra zeros are used to fill the least significant bits of the last byte of the binary value.

Example

`SELECT CAST( CAST( '1111' AS VARBIT ) AS BINARY )` returns 0xF0 (1111_2 becomes 11110000_2 = 0xF0).

`SELECT CAST( CAST( '0011000000110001' AS VARBIT ) AS BINARY )` returns 0x3031 (0011000000110001_2 = 0x3031).

Converting Bit Arrays to Characters

When converting a bit array of length `n` bits to a character data type, the length of the result is `n` characters. Each character in the result is either '0' or '1', corresponding to the bit in the array.

Example

`SELECT CAST( CAST( '01110' AS VARBIT ) AS VARCHAR )` returns the character string '01110'.

1.2.12.4 Numeric Set Conversions

When converting a DOUBLE type to a NUMERIC type, precision is maintained for the first 15 significant digits.

Related Information

CAST Function [Data Type Conversion] [page 275]
CONVERT Function [Data Type Conversion] [page 294]

1.2.12.5 Java and SQL Data Type Conversions

Data type conversion between Java types and SQL types is required for both Java stored procedures and JDBC applications. Java to SQL and SQL to Java data type conversions are carried out according to the JDBC standard. The conversions are described in the following tables.

In this section:

1.2.12.5.1 Java to SQL Data Type Conversions

Data type conversion between Java types and SQL types is required for both Java stored procedures and JDBC applications.

Java type	SQL type
String	CHAR
String	VARCHAR
String	TEXT
java.math.BigDecimal	NUMERIC
java.math.BigDecimal	MONEY
java.math.BigDecimal	SMALLMONEY
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[ ]	VARBINARY
byte[ ]	IMAGE
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.lang.Double	DOUBLE
java.lang.Float	REAL
java.lang.Integer	INTEGER
java.lang.Long	BIGINT

1.2.12.5.2 SQL to Java Data Type Conversions

Data type conversion between SQL types and Java types follow a specific mapping.

SQL type	Java type
CHAR	String
VARCHAR	String
TEXT	String

SQL type	Java type
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
MONEY	java.math.BigDecimal
SMALLMONEY	java.math.BigDecimal
UNSIGNED BIGINT	java.math.BigDecimal (precision=20, scale=0)
BIT	boolean
TINYINT	byte
SMALLINT	short
UNSIGNED SMALLINT	int
INTEGER	int
UNSIGNED INTEGER	long
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[ ]
VARBINARY	byte[ ]
LONG BINARY	byte[ ]
IMAGE	byte[ ]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

1.3 SQL Functions

Functions are used to return information from the database. They can be called anywhere an expression is allowed.

Unless otherwise specified in the documentation, NULL is returned for a function if any argument is NULL.

Functions use the same syntax conventions used by SQL statements.

In SQL Anywhere, if an argument is optional, then DEFAULT can be provided as an argument.

In this section:

Function Types [page 207]
Functions can be grouped according to the type of data they operate on, or the context in which they are used.

Functions [page 226]
Each function is listed, and the function type (numeric, character, and so on) is indicated next to it.

Related Information

Syntax Conventions [page 641]
Spatial Types and Functions

1.3.1 Function Types

Functions can be grouped according to the type of data they operate on, or the context in which they are used.

> **i** Note
>
> Unless otherwise stated, any SQL Anywhere function that receives NULL as a parameter returns NULL.

UltraLite: UltraLite supports a subset of the same functions documented for SQL Anywhere, and sometimes with a few differences.

In this section:

Aggregate Functions [page 208]
Aggregate functions summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. Aggregate functions are allowed only in the SELECT list and in the HAVING and ORDER BY clauses of a SELECT statement.

Composite Functions [page 210]
Composite functions allow you to perform tasks on arrays.

Bit Array Functions [page 210]
Bit array functions allow you to perform tasks on bit arrays.

Ranking Functions [page 211]
Ranking functions let you compute a rank value for each row in a result set based on an ordering specified in the query.

Data Type Conversion Functions [page 212]
Data type conversion functions are used to convert arguments from one data type to another, or to test whether they can be converted.

Date and Time Functions [page 213]
Date and time functions perform operations on DATE, TIME, TIMESTAMP, and TIMESTAMP WITH TIME ZONE data types.

Support for User-defined Functions [page 217]
A user-defined function, or UDF, is a function created by the user of a program or environment. User-defined functions are in contrast to functions that are built in to the program or environment.

Miscellaneous Functions [page 218]

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

Numeric Functions [page 219]
Numeric functions perform mathematical operations on numerical data types or return numeric information.

Web Services Functions [page 221]
HTTP functions assist the handling of HTTP requests within web services. Likewise, SOAP functions assist the handling of SOAP requests within web services.

String Functions [page 222]
String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

System Functions [page 224]
System functions return system information.

Text and Image Functions [page 226]
Text and image functions operate on text and image data types. Only the TEXTPTR text and image function is supported.

1.3.1.1 Aggregate Functions

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. Aggregate functions are allowed only in the SELECT list and in the HAVING and ORDER BY clauses of a SELECT statement.

List of SQL Anywhere Functions

The following aggregate functions are available:

ARRAY_AGG function [Aggregate]
AVG function [Aggregate]
BIT_AND function [Aggregate]
BIT_OR function [Aggregate]
BIT_XOR function [Aggregate]
COVAR_POP function [Aggregate]
COVAR_SAMP function [Aggregate]
COUNT function [Aggregate]
COUNT_BIG function [Aggregate]
CORR function [Aggregate]
FIRST_VALUE function [Aggregate]
GROUPING function [Aggregate]
LAST_VALUE function [Aggregate]
LIST function [Aggregate]
MAX function [Aggregate]

MEDIAN function [Aggregate]
MIN function [Aggregate]
REGR_AVGX function [Aggregate]
REGR_AVGY function [Aggregate]
REGR_COUNT function [Aggregate]
REGR_INTERCEPT function [Aggregate]
REGR_R2 function [Aggregate]
REGR_SLOPE function [Aggregate]
REGR_SXX function [Aggregate]
REGR_SXY function [Aggregate]
REGR_SYY function [Aggregate]
SET_BITS function [Aggregate]
STDDEV function [Aggregate]
STDDEV_POP function [Aggregate]
STDDEV_SAMP function [Aggregate]
SUM function [Aggregate]
VAR_POP function [Aggregate]
VAR_SAMP function [Aggregate]
VARIANCE function [Aggregate]
XMLAGG function [Aggregate]

List of UltraLite Functions

The following aggregate functions are available:

AVG function [Aggregate]
COUNT function [Aggregate]
COUNT_UPLOAD_ROWS function [Aggregate]
LIST function [Aggregate]
MAX function [Aggregate]
MIN function [Aggregate]
SUM function [Aggregate]

1.3.1.2 Composite Functions

Composite functions allow you to perform tasks on arrays.

List of Functions

The following composite functions are available:

ARRAY constructor [Composite]
ROW constructor [Composite]
ARRAY_MAX_CARDINALITY function [Composite]
CARDINALITY function [Composite]
TRIM_ARRAY function [Composite]

Related Information

ARRAY_AGG Function [Aggregate] [page 247]
UNNEST Array Operator [page 28]

1.3.1.3 Bit Array Functions

Bit array functions allow you to perform tasks on bit arrays.

List of Functions

The following bit array functions are available:

- BIT_AND function [Aggregate]
- BIT_OR function [Aggregate]
- BIT_XOR function [Aggregate]
- BIT_LENGTH function [Bit array]
- BIT_SUBSTR function [Bit array]
- COUNT_SET_BITS function [Bit array]
- GET_BIT function [Bit array]
- SET_BIT function [Bit array]
- SET_BITS function [Aggregate]

Related Information

Bitwise Operators [page 33]
sa_get_bits System Procedure [page 1596]
BIT_AND Function [Aggregate] [page 260]
BIT_OR Function [Aggregate] [page 262]
BIT_XOR Function [Aggregate] [page 265]
BIT_LENGTH Function [Bit Array] [page 261]
BIT_SUBSTR Function [Bit Array] [page 263]
COUNT_SET_BITS Function [Bit Array] [page 304]
GET_BIT Function [Bit Array] [page 389]
SET_BIT Function [Bit Array] [page 548]
SET_BITS Function [Aggregate] [page 549]

1.3.1.4 Ranking Functions

Ranking functions let you compute a rank value for each row in a result set based on an ordering specified in the query.

List of Functions

The following rank functions are available:

CUME_DIST function [Ranking]
DENSE_RANK function [Ranking]
PERCENT_RANK function [Ranking]
RANK function [Ranking]

Related Information

CUME_DIST Function [Ranking] [page 311]
DENSE_RANK Function [Ranking] [page 346]
PERCENT_RANK Function [Ranking] [page 485]
RANK Function [Ranking] [page 501]

1.3.1.5 Data Type Conversion Functions

Data type conversion functions are used to convert arguments from one data type to another, or to test whether they can be converted.

List of SQL Anywhere Functions

The following data type conversion functions are available:

BINTOHEX Function [Data Type Conversion]
CAST Function [Data Type Conversion]
CONVERT Function [Data Type Conversion]
HEXTOBIN Function [Data Type Conversion]
HEXTOINT Function [Data Type Conversion]
INTTOHEX Function [Data Type Conversion]
ISDATE Function [Data Type Conversion]
ISNUMERIC Function [Miscellaneous]
TREAT Function [Data Type Conversion]

List of UltraLite Functions

The following data type conversion functions are available:

CAST Function [Data Type Conversion]
CONVERT Function [Data Type Conversion]
HEXTOINT Function [Data Type Conversion]
INTTOHEX Function [Data Type Conversion]
ISDATE Function [Data Type Conversion]

Related Information

BINTOHEX Function [Data Type Conversion] [page 258]
CAST Function [Data Type Conversion] [page 275]
CONVERT Function [Data Type Conversion] [page 294]
HEXTOBIN Function [Data Type Conversion] [page 399]
HEXTOINT Function [Data Type Conversion] [page 400]
INTTOHEX Function [Data Type Conversion] [page 424]
ISDATE Function [Data Type Conversion] [page 425]
ISNUMERIC Function [Miscellaneous] [page 430]
TREAT Function [Data Type Conversion] [page 600]

1.3.1.6 Date and Time Functions

Date and time functions perform operations on DATE, TIME, TIMESTAMP, and TIMESTAMP WITH TIME ZONE data types.

SQL Anywhere includes compatibility support for Transact-SQL date and time types, including DATETIME and SMALLDATETIME. These Transact-SQL data types are implemented as domains over the native TIMESTAMP data type.

The following date and time functions are available for SQL Anywhere:

DATE function [Date and time]
DATEADD function [Date and time]
DATEDIFF function [Date and time]
DATEFORMAT function [Date and time]
DATENAME function [Date and time]
DATEPART function [Date and time]
DATETIME function [Date and time]
DAY function [Date and time]
DAYNAME function [Date and time]
DAYS function [Date and time]
DOW function [Date and time]
GETDATE function [Date and time]
HOUR function [Date and time]
HOURS function [Date and time]
MINUTE function [Date and time]
MINUTES function [Date and time]
MONTH function [Date and time]
MONTHNAME function [Date and time]
MONTHS function [Date and time]
NOW function [Date and time]
QUARTER function [Date and time]
SECOND function [Date and time]
SECONDS function [Date and time]
SWITCHOFFSET function [Date and time]
SYSDATETIMEOFFSET function [Date and time]
TODAY function [Date and time]
TODATETIMEOFFSET function [Date and time]
WEEKS function [Date and time]
YEAR function [Date and time]
YEARS function [Date and time]
YMD function [Date and time]

The following date and time functions are available for UltraLite:

DATE function [Date and time]
DATEADD function [Date and time]
DATEDIFF function [Date and time]

DATEFORMAT function [Date and time]
DATENAME function [Date and time]
DATEPART function [Date and time]
DATETIME function [Date and time]
DAY function [Date and time]
DAYNAME function [Date and time]
DAYS function [Date and time]
DOW function [Date and time]
GETDATE function [Date and time]
HOUR function [Date and time]
HOURS function [Date and time]
MINUTE function [Date and time]
MINUTES function [Date and time]
MONTH function [Date and time]
MONTHNAME function [Date and time]
MONTHS function [Date and time]
NOW function [Date and time]
QUARTER function [Date and time]
SECOND function [Date and time]
SECONDS function [Date and time]
SWITCHOFFSET function [Date and time]
TODAY function [Date and time]
TODATETIMEOFFSET function [Date and time]
WEEKS function [Date and time]
YEAR function [Date and time]
YEARS function [Date and time]
YMD function [Date and time]

In this section:

1.3.1.6.1 Specifying Date Parts

Many of the date functions use dates built from **date parts**. The following table displays the allowed date part specifiers, their short forms, and the range of values returned by the DATEPART function.

Date part	Abbreviation	Values
Year	YY	1-9999
Quarter	QQ	1-4
Month	MM	1-12

Date part	Abbreviation	Values
Week	WK	1-54. Weeks begin on Sunday. A 54-week year occurs in leap years that start on a Saturday. Week is not subject to the first_day_of_week setting.
Day	DD	1-31
Dayofyear	DY	1-366
Weekday	DW	1-7. Weekday is subject to the first_day_of_week setting. For example, If the first day of week is Monday, then Monday is 1 and Sunday is 7.
Hour	HH	0-23
Minute	MI	0-59
Second	SS	0-59
Millisecond	MS	0-999
Microsecond	MCS or US	0-999999
Calyearofweek	CYR	1-9999. The year in which the week begins. The ISO standard first full week of any year always begins on a Monday. The first week of the year can start before, on, or after the first day of the year. If at least the first 4 days of the year occur in a week, that week is considered to be the first week of the year. Any days of the previous calendar year, that also fall in the first week of the year, are included. Otherwise, the next week is the first full week of the year. In this case, any days of the previous week are part of the last full week of the previous year.
Calweekofyear	CWK	1-53. The week number within the year that contains the specified date. For more information about the ISO week system and the ISO 8601 date and time standard, see ISO week date .
Caldayofweek	CDW	1-7. (Monday = 1, ..., Sunday = 7)
TZOffset	TZ	-840 to 840

Note that Sunday is the last day of the week in the ISO 8601 calendar, whereas Sunday is considered the first day of the week in some locales (for example, the United States, Canada, and Japan).

Calyearofweek, Calweekofyear, and Caldayofweek conform to ISO 8601 in which weeks start with Monday. The first week of a year is the week that contains the first Thursday of the year (and, hence, always contains 4 January). These values are not affected by the first_day_of_week option setting.

The ISO standard numbers each weekday as follows: Monday=1, Tuesday=2, ..., Sunday=7. To calculate the first Monday of the year, the week is split into two groups. The first, major group contains the 4 days Monday to Thursday. The second, minor group contains the 3 days Friday to Sunday.

If the first day of the year falls in the first group (Monday to Thursday), then the majority of the days in the week fall in this year and all days of that week including days that are part of the previous year are considered to belong to the first full week of this year. For example, Monday 2014-12-29 occurs in the first full week of 2015 because 2015-01-01 is a Thursday (the majority of the days in that week are part of 2015). Here Calyearofweek (CYR) for those days is 2015.

```
     January 2015
 M  T  W  T  F  S  S  CYR 2015
29 30 31  1  2  3  4  CYR 2015
 5  6  7  8  9 10 11  CYR 2015
12 13 14 15 16 17 18  CYR 2015
19 20 21 22 23 24 25  CYR 2015
26 27 28 29 30 31     CYR 2015
```

If the first day of the year falls in the second group (Friday to Sunday), then the first Monday of the year falls in the next week. In this case, the first few days of the year before that Monday are considered to fall in the last full week of the previous year. For example, Friday 2016-01-01 to Sunday 2016-01-03 fall in the last full week of 2015 (the majority of the days in that week are part of 2015). Here Calyearofweek (CYR) for those days is 2015, not 2016.

```
     January 2016
 M  T  W  T  F  S  S
28 29 30 31  1  2  3  CYR 2015
 4  5  6  7  8  9 10  CYR 2016
11 12 13 14 15 16 17  CYR 2016
18 19 20 21 22 23 24  CYR 2016
25 26 27 28 29 30 31  CYR 2016
```

Related Information

Date and Time Data Types [page 160]
Functions [page 226]

1.3.1.7 Support for User-defined Functions

A user-defined function, or UDF, is a function created by the user of a program or environment. User-defined functions are in contrast to functions that are built in to the program or environment.

User-Defined Functions in SQL

You can implement your own functions in SQL by using the CREATE FUNCTION statement.

The RETURN statement inside the CREATE FUNCTION statement determines the data type of the function.

Once a SQL user-defined function is created, it can be used anywhere a built-in function of the same data type is used.

User-Defined Functions in Java and the CLR

Java classes provide a more powerful and flexible way of implementing user-defined functions, with the additional advantage that they can be moved from the database server to a client application if desired. Any class method of an installed Java class can be used as a user-defined function anywhere a built-in function of the same data type is used. Instance methods are tied to particular instances of a class, and so have different behavior from standard user-defined functions.

Support for CLR stored procedures and functions is included. A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a Microsoft .NET language such as Microsoft C# or Microsoft Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate Microsoft .NET executable). Only Microsoft .NET version 2.0 is supported.

Deciding Whether to Create a User-Defined Function or Procedure

Functions are similar to procedures. Deciding whether to create a function or a procedure depends on what you want returned, and the object will be called. When deciding whether to create a UDF or a procedure, consider their unique characteristics listed below.

Functions:

- can return a single value of arbitrary type, and allow you to declare the returned type using the RETURNS clause
- can be used in most places an expression can be used
- allow you to define only IN parameters

Procedures:

- can return multiple values using INOUT or OUT parameters

- can return result sets
- can be referenced in the FROM clause of a query, or using a CALL statement, or using a Transact-SQL EXECUTE statement
- can be called using named parameters

Related Information

Class File Creation
Stored Procedures, Triggers, Batches, and User-defined Functions
The CLR External Environment
CREATE FUNCTION Statement [page 861]
Named Parameters [page 128]

1.3.1.8 Miscellaneous Functions

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

List of SQL Anywhere Functions

The following miscellaneous functions are available:

ARGN Function [Miscellaneous]
COALESCE Function [Miscellaneous]
CONFLICT Function [Miscellaneous]
ERRORMSG Function [Miscellaneous]
ESTIMATE Function [Miscellaneous]
ESTIMATE_SOURCE Function [Miscellaneous]
EXPERIENCE_ESTIMATE Function [Miscellaneous]
EXPLANATION Function [Miscellaneous]
EXPRTYPE Function [Miscellaneous]
GET_IDENTITY Function [Miscellaneous]
GRAPHICAL_PLAN Function [Miscellaneous]
GREATER Function [Miscellaneous]
IDENTITY Function [Miscellaneous]
IFNULL Function [Miscellaneous]
INDEX_ESTIMATE Function [Miscellaneous]
ISNULL Function [Miscellaneous]
LESSER Function [Miscellaneous]
NEWID Function [Miscellaneous]

NULLIF Function [Miscellaneous]
NUMBER Function [Miscellaneous]
PLAN Function [Miscellaneous]
REWRITE Function [Miscellaneous]
ROW_NUMBER Function [Miscellaneous]
SQLDIALECT Function [Miscellaneous]
SQLFLAGGER Function [Miscellaneous]
ERROR_LINE Function [Miscellaneous]
TRACEBACK Function [Miscellaneous]
TRANSACTSQL Function [Miscellaneous]
VAREXISTS Function [Miscellaneous]
WATCOMSQL Function [Miscellaneous]

List of UltraLite Functions

The following miscellaneous functions are available:

ARGN Function [Miscellaneous]
COALESCE Function [Miscellaneous]
EXPLANATION Function [Miscellaneous]
GREATER Function [Miscellaneous]
IFNULL Function [Miscellaneous]
ISNULL Function [Miscellaneous]
LESSER Function [Miscellaneous]
NEWID Function [Miscellaneous]
NULLIF Function [Miscellaneous]

1.3.1.9 Numeric Functions

Numeric functions perform mathematical operations on numerical data types or return numeric information.

List of SQL Anywhere Functions

The following numeric functions are available:

ABS function [Numeric]
ACOS function [Numeric]
ASIN function [Numeric]
ATAN function [Numeric]
ATAN2 function [Numeric]
CEILING function [Numeric]

COS function [Numeric]
COT function [Numeric]
DEGREES function [Numeric]
EXP function [Numeric]
FLOOR function [Numeric]
LOG function [Numeric]
LOG10 function [Numeric]
MOD function [Numeric]
PI function [Numeric]
POWER function [Numeric]
RADIANS function [Numeric]
RAND function [Numeric]
REMAINDER function [Numeric]
ROUND function [Numeric]
SIGN function [Numeric]
SIN function [Numeric]
SQRT function [Numeric]
TAN function [Numeric]
TRUNCNUM function [Numeric]

List of UltraLite Functions

The following numeric functions are available:

ABS function [Numeric]
ACOS function [Numeric]
ASIN function [Numeric]
ATAN function [Numeric]
ATAN2 function [Numeric]
CEILING function [Numeric]
COS function [Numeric]
COT function [Numeric]
DEGREES function [Numeric]
EXP function [Numeric]
FLOOR function [Numeric]
LOG function [Numeric]
LOG10 function [Numeric]
MOD function [Numeric]
PI function [Numeric]
POWER function [Numeric]
RADIANS function [Numeric]
REMAINDER function [Numeric]
ROUND function [Numeric]
SIGN function [Numeric]

SIN function [Numeric]
SQRT function [Numeric]
TAN function [Numeric]
TRUNCNUM function [Numeric]

1.3.1.10 Web Services Functions

HTTP functions assist the handling of HTTP requests within web services. Likewise, SOAP functions assist the handling of SOAP requests within web services.

The following functions are available:

- HTML_DECODE Function [Miscellaneous]
- HTML_ENCODE Function [Miscellaneous]
- HTTP_BODY function [Web service]
- HTTP_DECODE function [Web service]
- HTTP_ENCODE function [Web service]
- HTTP_HEADER function [Web service]
- HTTP_RESPONSE_HEADER function [Web service]
- HTTP_VARIABLE function [Web service]
- NEXT_HTTP_HEADER function [Web service]
- NEXT_HTTP_RESPONSE_HEADER function [Web service]
- NEXT_HTTP_VARIABLE function [Web service]
- NEXT_SOAP_HEADER function [SOAP]
- SOAP_HEADER function [SOAP]

There are also system procedures available for web services.

Related Information

The Database Server as an HTTP Web Server
-xs Database Server Option
Web Services System Procedures [page 1512]
HTML_DECODE Function [Miscellaneous] [page 405]
HTML_ENCODE Function [Miscellaneous] [page 407]
HTTP_BODY Function [Web Service] [page 408]
HTTP_DECODE Function [Web Service] [page 409]
HTTP_ENCODE Function [Web Service] [page 410]
HTTP_HEADER Function [Web Service] [page 412]
HTTP_RESPONSE_HEADER Function [Web Service] [page 414]
HTTP_VARIABLE Function [Web Service] [page 417]
NEXT_HTTP_HEADER Function [Web Service] [page 472]
NEXT_HTTP_RESPONSE_HEADER Function [Web Service] [page 474]

NEXT_HTTP_VARIABLE Function [Web Service] [page 475]
NEXT_SOAP_HEADER Function [SOAP] [page 477]
SOAP_HEADER Function [SOAP] [page 554]

1.3.1.11 String Functions

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

When working in a multibyte character set, check carefully whether the function being used returns information concerning characters or bytes.

List of SQL Anywhere Functions

The following string functions are available:

ASCII function [String]
BASE64_DECODE function [String]
BASE64_ENCODE function [String]
BYTE_LENGTH function [String]
BYTE_SUBSTR function [String]
CHAR function [String]
CHARINDEX function [String]
CHAR_LENGTH function [String]
COMPARE function [String]
COMPRESS function [String]
CSCONVERT function [String]
DECOMPRESS function [String]
DECRYPT function [String]
DIFFERENCE function [String]
ENCRYPT function [String]
HASH function [String]
INSERTSTR function [String]
LCASE function [String]
LEFT function [String]
LENGTH function [String]
LOCATE function [String]
LOWER function [String]
LTRIM function [String]
NCHAR function [String]
PATINDEX function [String]
READ_CLIENT_FILE function [String]
READ_SERVER_FILE function [String]

REGEXP_SUBSTR function [String]
REPEAT function [String]
REPLACE function [String]
REPLICATE function [String]
REVERSE function [String]
RIGHT function [String]
RTRIM function [String]
SIMILAR function [String]
SORTKEY function [String]
SOUNDEX function [String]
SPACE function [String]
STR function [String]
STRING function [String]
STRTOUUID function [String]
STUFF function [String]
SUBSTRING function [String]
TO_CHAR function [String]
TO_NCHAR function [String]
TRIM function [String]
UCASE function [String]
UNICODE function [String]
UNISTR function [String]
UPPER function [String]
UUIDTOSTR function [String]
XMLCONCAT function [String]
XMLELEMENT function [String]
XMLFOREST function [String]
XMLGEN function [String]

List of UltraLite Functions

The following string functions are available:

ASCII function [String]
BYTE_LENGTH function [String]
BYTE_SUBSTR function [String]
CHAR function [String]
CHARINDEX function [String]
CHAR_LENGTH function [String]
DIFFERENCE function [String]
INSERTSTR function [String]
LCASE function [String]
LEFT function [String]
LENGTH function [String]

LOCATE function [String]
LOWER function [String]
LTRIM function [String]
PATINDEX function [String]
REPEAT function [String]
REPLACE function [String]
REPLICATE function [String]
RIGHT function [String]
RTRIM function [String]
SIMILAR function [String]
SOUNDEX function [String]
SPACE function [String]
STR function [String]
STRING function [String]
STRTOUUID function [String]
STUFF function [String]
SUBSTRING function [String]
TRIM function [String]
UCASE function [String]
UPPER function [String]
UUIDTOSTR function [String]

1.3.1.12 System Functions

System functions return system information.

List of Functions

The following system functions are available:

CONNECTION_EXTENDED_PROPERTY function [String]
CONNECTION_PROPERTY function [System]
DATALENGTH function [System]
DB_ID function [System]
DB_NAME function [System]
DB_EXTENDED_PROPERTY function [System]
DB_PROPERTY function [System]
EVENT_CONDITION function [System]
EVENT_CONDITION_NAME function [System]
EVENT_PARAMETER function [System]
NEXT_CONNECTION function [System]
NEXT_DATABASE function [System]

PROPERTY function [System]
PROPERTY_DESCRIPTION function [System]
PROPERTY_NAME function [System]
PROPERTY_NUMBER function [System]
SUSER_ID function [System]
SUSER_NAME function [System]
TSEQUAL function [System] (deprecated)
USER_ID function [System]
USER_NAME function [System]
DB_PROPERTY function [System]

UltraLite Functions

DB_PROPERTY function [System]
ML_GET_SERVER_NOTIFICATION function [System]
SYNC_PROFILE_OPTION_VALUE function [System]

SQL Anywhere notes

- The db_id, db_name, and datalength functions are implemented as built-in functions.
- Some system functions are implemented as stored procedures.

System functions that are not described elsewhere are noted in the following table. These functions are implemented as stored procedures.

Syntax: COL_LENGTH

```
COL_LENGTH( @object_name, @column_name )
```

Returns the INTEGER defined length of the specified column. `@object_name` can contain the owner, for example, 'GROUPO.Customers'.

Syntax: COL_TERM

```
COL_NAME( @object_id, @column_id [, @database_id ] )
```

Returns the CHAR(128) column name.

Syntax: INDEX_COL

```
INDEX_COL ( @table_name, @index_id, @key_# [, @user_id ] )
```

Returns the CHAR(128) name of the indexed column. `@table_name` can contain the owner, for example, 'GROUPO.Customers'.

Syntax: OBJECT_ID

```
OBJECT_ID( @object_name )
```

Returns the INTEGER object ID. `@object_name` can contain the owner, for example, 'GROUPO.Customers'.

Syntax: OBJECT_NAME

```
OBJECT_NAME ( @object_id [, @database_id ] )
```

Returns the CHAR(128) object name.

1.3.1.13 Text and Image Functions

Text and image functions operate on text and image data types. Only the TEXTPTR text and image function is supported.

1.3.2 Functions

Each function is listed, and the function type (numeric, character, and so on) is indicated next to it.

In this section:

Decodes data using the MIME base64 format and returns the string as a LONG VARCHAR.

BASE64_ENCODE Function [String] [page 257]
Encodes data using the MIME base64 format and returns it as a 7-bit ASCII string.

BINTOHEX Function [Data Type Conversion] [page 258]
Returns the hexadecimal equivalent of a binary string.

BIT_AND Function [Aggregate] [page 260]
Returns the bit-wise AND of the specified expression for each group of rows.

BIT_LENGTH Function [Bit Array] [page 261]
Returns the number of bits stored in the array.

BIT_OR Function [Aggregate] [page 262]
Returns the bit-wise OR of the specified expression for each group of rows.

BIT_SUBSTR Function [Bit Array] [page 263]
Returns a sub-array of a bit array.

BIT_XOR Function [Aggregate] [page 265]
Returns the bit-wise XOR of the specified expression for each group of rows.

BYTE_INSERTSTR Function [String] [page 266]
Inserts a string into another string at a position specified in bytes.

BYTE_LENGTH Function [String] [page 267]
Returns the number of bytes in a string.

BYTE_LOCATE Function [String] [page 268]
Returns the position of one BYTE string within another.

BYTE_REPLACE Function [String] [page 270]
Replaces a string with another string, and returns the new result.

BYTE_STUFF Function [String] [page 271]
Deletes multiple bytes from one string and replaces them with different bytes.

BYTE_SUBSTR Function [String] [page 272]
Returns a substring of a string. The substring is determined using bytes, not characters.

CARDINALITY Function [Composite] [page 274]
Returns the highest number of any array element that has been assigned a value, including NULL.

CAST Function [Data Type Conversion] [page 275]
Returns the value of an expression converted to a supplied data type.

CEILING Function [Numeric] [page 278]
Returns the first integer that is greater or equal to a given value. For positive numbers, this is known as rounding up.

CHAR Function [String] [page 279]
Returns the character with the ASCII value of a number.

CHAR_LENGTH Function [String] [page 280]
Returns the number of characters in a string.

CHARINDEX Function [String] [page 282]
Returns the position of one string in another.

COALESCE Function [Miscellaneous] [page 283]
Returns the first non-NULL expression from a list. This function is identical to the ISNULL function.

Returns the interval between two dates.

DATEFORMAT Function [Date and Time] [page 319]
Returns a string representing a date expression in the specified format.

DATENAME Function [Date and Time] [page 321]
Returns the name of the specified part (such as the month June) of a TIMESTAMP or TIMESTAMP WITH TIME ZONE value, as a character string.

DATEPART Function [Date and Time] [page 322]
Returns a portion of a TIMESTAMP or TIMESTAMP WITH TIME ZONE value.

DATETIME Function [Date and Time] [page 324]
Converts an expression into a TIMESTAMP value.

DAY Function [Date and Time] [page 325]
Returns the day of the month of its argument as an integer between 1 and 31.

DAYNAME Function [Date and Time] [page 326]
Returns the name of the day of the week from a date.

DAYS Function [Date and Time] [page 327]
Manipulates a TIMESTAMP or returns the number of days between two TIMESTAMP values.

DB_EXTENDED_PROPERTY Function [System] [page 329]
Returns the value of the given property. Allows an optional property-specific string parameter to be specified.

DB_ID Function [System] [page 334]
Returns the database ID number.

DB_NAME Function [System] [page 336]
Returns the name of a database with a given ID number.

DB_PROPERTY Function [System] [page 337]
Returns the value of the specified database property.

DECOMPRESS Function [String] [page 339]
Decompresses the string and returns a LONG BINARY value.

DECRYPT Function [String] [page 340]
Decrypts the string using the supplied key and returns a LONG BINARY value.

DEGREES Function [Numeric] [page 345]
Converts a number from radians to degrees.

DENSE_RANK Function [Ranking] [page 346]
Calculates the rank of a value in a partition. For tied values, the DENSE_RANK function does not leave gaps in the ranking sequence.

DIFFERENCE Function [String] [page 348]
Returns the difference in the SOUNDEX values between the two string expressions.

DOW Function [Date and Time] [page 349]
Returns a number from 1 to 7 representing the day of the week of a date, where Sunday=1, Monday=2, and so on.

ENCRYPT Function [String] [page 350]
Encrypts the specified value using the supplied encryption key and returns a LONG BINARY value.

ERROR_LINE Function [Miscellaneous] [page 355]

Returns the line number of the procedure or batch where the error occurred that invoked the CATCH block of a TRY...CATCH statement.

ERROR_MESSAGE Function [Miscellaneous] [page 356]
Returns the message text of the error that invoked the CATCH block of a TRY...CATCH statement.

ERROR_PROCEDURE Function [Miscellaneous] [page 358]
Returns the name of the procedure within which the error that caused the exception handler to run occurred.

ERROR_SQLCODE Function [Miscellaneous] [page 359]
Returns the SQLCODE of the error that invoked the error handler.

ERROR_SQLSTATE Function [Miscellaneous] [page 361]
Returns the SQLSTATE of the error that invoked the error handler.

ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
Returns a calling sequence stack trace for the error that invoked the error handler.

ERRORMSG Function [Miscellaneous] [page 364]
Provides the error message for the current error, or for a specified SQLSTATE or SQLCODE value.

ESTIMATE Function [Miscellaneous] [page 366]
Returns selectivity estimates as a percentage calculated by the query optimizer, based on specified parameters.

ESTIMATE_SOURCE Function [Miscellaneous] [page 367]
Provides the source for selectivity estimates used by the query optimizer.

EVENT_CONDITION Function [System] [page 369]
Specifies when an event handler is triggered.

EVENT_CONDITION_NAME Function [System] [page 371]
Lists the possible parameters for EVENT_CONDITION.

EVENT_PARAMETER Function [System] [page 372]
Provides context information for event handlers.

EXP Function [Numeric] [page 375]
Returns the result of the base of natural logarithms e raised to the power of the given argument.

EXPERIENCE_ESTIMATE Function [Miscellaneous] [page 376]
Returns selectivity estimates as a percentage calculated by the query optimizer, based on specified parameters.

EXPLANATION Function [Miscellaneous] [page 378]
Returns the optimization strategy of a SQL statement as a plain text string.

EXPRTYPE Function [Miscellaneous] [page 379]
Returns a string that identifies the data type of an expression.

EXTENDED_PROPERTY Function [System] [page 381]
Returns the value of the given database server property. Allows an optional property-specific string parameter to be specified.

EXTRACT Function [Date and Time] [page 383]
Returns a date part from a DATE, TIME, TIMESTAMP, or TIMESTAMP WITH TIME ZONE expression.

FIRST_VALUE Function [Aggregate] [page 385]
Returns values from the first row of a window.

Returns the current date and time as a TIMESTAMP value. The accuracy is limited by the accuracy of the system clock.

NULLIF Function [Miscellaneous] [page 480]
Provides an abbreviated CASE expression by comparing expressions.

NUMBER Function [Miscellaneous] [page 481]
Generates numbers starting at 1 for each successive row in the results of the query. The NUMBER function is primarily intended for use in SELECT lists.

PATINDEX Function [String] [page 483]
Returns an integer representing the starting position of the first occurrence of a pattern in a string.

PERCENT_RANK Function [Ranking] [page 485]
For any row X, defined by the function's arguments and ORDER BY specification, the PERCENT_RANK function determines the rank of row X - 1, divided by the number of rows in the group.

PI Function [Numeric] [page 487]
Returns the numeric value PI.

PLAN Function [Miscellaneous] [page 488]
Returns the long plan optimization strategy of a SQL statement, as a string.

POWER Function [Numeric] [page 489]
Calculates one number raised to the power of another.

PROPERTY Function [System] [page 490]
Returns the value of the specified database server property as a string.

PROPERTY_DESCRIPTION Function [System] [page 492]
Returns a description of a property.

PROPERTY_IS_TRACKABLE Function [System] [page 494]
Returns whether or not you can maintain historical data for the specified database server property by storing its tracked values.

PROPERTY_NAME Function [System] [page 495]
Returns the name of the property with the supplied property ID for the specified connection level.

PROPERTY_NUMBER Function [System] [page 496]
Returns the property number of the property with the supplied property-name.

QUARTER Function [Date and Time] [page 497]
Returns a number indicating the quarter of the year from the supplied TIMESTAMP expression.

RADIANS Function [Numeric] [page 498]
Converts a number from degrees to radians.

RAND Function [Numeric] [page 499]
Returns a random number in the interval 0 to 1, with an optional seed.

RANK Function [Ranking] [page 501]
Calculates the value of a rank in a group of values. For ties, the RANK function leaves a gap in the ranking sequence.

READ_CLIENT_FILE Function [String] [page 503]
Reads data from the specified file on the client computer.

READ_SERVER_FILE Function [String] [page 504]

Reads data from the specified file on the server and returns the full or partial contents of the file as a LONG BINARY value.

REGEXP_SUBSTR Function [String] [page 506]
Extracts substrings from strings using regular expressions.

REGR_AVGX Function [Aggregate] [page 508]
Computes the average of the independent variable of the regression line.

REGR_AVGY Function [Aggregate] [page 510]
Computes the average of the dependent variable of the regression line.

REGR_COUNT Function [Aggregate] [page 512]
Returns an integer that represents the number of non-NULL number pairs used to fit the regression line.

REGR_INTERCEPT Function [Aggregate] [page 513]
Computes the y-intercept of the linear regression line that best fits the dependent and independent variables.

REGR_R2 Function [Aggregate] [page 515]
Computes the coefficient of determination (also referred to as *R-squared* or the *goodness of fit* statistic) for the regression line.

REGR_SLOPE Function [Aggregate] [page 516]
Computes the slope of the linear regression line fitted to non-NULL pairs.

REGR_SXX Function [Aggregate] [page 518]
Returns the sum of squares of the independent expressions used in a linear regression model. The REGR_SXX function can be used to evaluate the statistical validity of a regression model.

REGR_SXY Function [Aggregate] [page 520]
Returns the sum of products of the dependent and independent variables. The REGR_SXY function can be used to evaluate the statistical validity of a regression model.

REGR_SYY Function [Aggregate] [page 521]
Returns values that can evaluate the statistical validity of a regression model.

REMAINDER Function [Numeric] [page 523]
Returns the remainder when one whole number is divided by another.

REPEAT Function [String] [page 524]
Concatenates a string a specified number of times.

REPLACE Function [String] [page 525]
Replaces a string with another string, and returns the new results.

REPLICATE Function [String] [page 527]
Concatenates a string a specified number of times.

REVERSE Function [String] [page 529]
Returns the reverse of a character expression.

REWRITE Function [Miscellaneous] [page 530]
Returns a rewritten SELECT, UPDATE, or DELETE statement.

RIGHT Function [String] [page 532]
Returns the rightmost characters of a string.

ROUND Function [Numeric] [page 533]

Rounds the `numeric-expression` to the specified `integer-expression` amount of places after the decimal point.

ROW Constructor [Composite] [page 534]
Returns a sequence of (`field name data type`, ...) pairs named **fields**.

ROW_NUMBER Function [Miscellaneous] [page 537]
Assigns a unique number to each row. Use this function instead of the NUMBER function.

ROWID Function [Miscellaneous] [page 538]
Returns an UNSIGNED BIGINT value that uniquely identifies a row within a table.

RTRIM Function [String] [page 540]
Removes trailing blanks or specified characters from the string.

SECOND Function [Date and Time] [page 542]
Returns the seconds value of the TIMESTAMP argument.

SECONDS Function [Date and Time] [page 543]
Manipulates a TIMESTAMP or returns the number of second boundaries between two TIMESTAMP values.

SECURE_SIGN_MESSAGE Function [String] [page 545]
Digitally signs a message.

SECURE_VERIFY_MESSAGE Function [String] [page 546]
Digitally verifies a message.

SET_BIT Function [Bit Array] [page 548]
Sets the value of a specific bit in a bit array.

SET_BITS Function [Aggregate] [page 549]
Creates a bit array where specific bits, corresponding to values from a set of rows, are set to 1 (TRUE).

SIGN Function [Numeric] [page 551]
Returns the sign (positive or negative) of the given number.

SIMILAR Function [String] [page 552]
Returns a number indicating the similarity between two strings.

SIN Function [Numeric] [page 553]
Returns the sine of a number.

SOAP_HEADER Function [SOAP] [page 554]
Returns a SOAP header entry, or an attribute value for a header entry of the SOAP request.

SORTKEY Function [String] [page 556]
Generates sort key values. That is, values that can be used to sort character strings based on alternate collation rules.

SOUNDEX Function [String] [page 559]
Returns a number representing the sound of a string.

SPACE Function [String] [page 561]
Returns a specified number of spaces.

SQLDIALECT Function [Miscellaneous] [page 562]
Returns either Watcom SQL or Transact-SQL, to indicate the SQL dialect of a statement.

SQLFLAGGER Function [Miscellaneous] [page 563]

Returns the conformity of a given SQL statement to a specified standard such as the ANSI/ISO SQL Standard.

SQRT Function [Numeric] [page 565]
Returns the square root of a number.

STACK_TRACE Function [Miscellaneous] [page 566]
Returns information about the stack trace for the current statement.

STDDEV Function [Aggregate] [page 568]
An alias for STDDEV_SAMP.

STDDEV_POP Function [Aggregate] [page 568]
Computes the standard deviation of a population consisting of a numeric-expression, as a DOUBLE.

STDDEV_SAMP Function [Aggregate] [page 570]
Computes the standard deviation of a sample consisting of a numeric-expression, as a DOUBLE.

STR Function [String] [page 573]
Returns the string equivalent of a number.

STRING Function [String] [page 574]
Concatenates one or more strings into one large string.

STRTOUUID Function [String] [page 575]
Converts a string value to a unique identifier (UUID or GUID) value.

STUFF Function [String] [page 577]
Deletes multiple characters from one string and replaces them with another string.

SUBSTRING Function [String] [page 578]
Returns a substring of a string.

SUM Function [Aggregate] [page 581]
Returns the total of the specified expression for each group of rows.

SUSER_ID Function [System] [page 583]
Returns the numeric user ID for the specified user name.

SUSER_NAME Function [System] [page 584]
Returns the user name for the specified user ID.

SWITCHOFFSET Function [Date and Time] [page 585]
Returns a TIMESTAMP WITH TIME ZONE value that is converted from its original time zone offset to the specified time zone offset.

SYSDATETIMEOFFSET Function [Date and Time] [page 586]
Returns the current date, time, and time zone offset of the database server using the system clock.

TAN Function [Numeric] [page 587]
Returns the tangent of a number.

TEXTPTR Function [Text and Image] [page 588]
Returns a 16-byte binary pointer to the specified column. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

TO_CHAR Function [String] [page 589]
Converts character data from any supported character set into the CHAR character set for the database.

TO_NCHAR Function [String] [page 591]

Converts character data from any supported character set into the NCHAR character set.

TODATETIMEOFFSET Function [Date and Time] [page 592]
Converts a TIMESTAMP value to a TIME STAMP WITH TIME ZONE value using the specified time zone offset.

TODAY Function [Date and Time] [page 593]
Returns the current date as a DATE value.

TOLOCALTIME Function [Date and time] [page 594]
Converts a TIMESTAMP WITH TIME ZONE value, or a TIMESTAMP value (which is assumed to be in Coordinated Universal Time (UTC)), to a timestamp value that corresponds to the server's local time using the standard time or daylight saving time rules for the server's locale.

TRACEBACK Function [Miscellaneous] [page 596]
Returns statements on the stack of the most recent exception (error) that occurred during a stored procedure, trigger, or custom function execution.

TRACED_PLAN Function [Miscellaneous] (Deprecated) [page 598]
The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. This function is used by *SQL Central* to generate a graphical plan for a query using tracing data.

TRANSACTSQL Function [Miscellaneous] [page 599]
Rewrites a Watcom SQL statement in the Transact-SQL dialect.

TREAT Function [Data Type Conversion] [page 600]
Changes the declared type of a geometry expression to a subtype. This function is for use with spatial data.

TRIM Function [String] [page 601]
Removes leading and trailing blanks or specified characters from a string.

TRIM_ARRAY Function [Composite] [page 603]
Returns an implicitly bounded array that consists of a specified number of elements in an array.

TRUNCNUM Function [Numeric] [page 604]
Truncates a number at a specified number of places after the decimal point.

TSEQUAL function [System] (Deprecated) [page 606]
Compares two TIMESTAMP values and returns whether they are the same.

UCASE Function [String] [page 607]
Converts all characters in a string to uppercase.

UNICODE Function [String] [page 608]
Returns an integer containing the Unicode code point of the first character in the string, or NULL if the first character is not a valid encoding.

UNISTR Function [String] [page 609]
Converts a string containing characters and Unicode escape sequences to an NCHAR string.

UPPER Function [String] [page 611]
Converts all characters in a string to uppercase.

USER_ID Function [System] [page 612]
Returns the numeric user ID for the specified user name.

USER_NAME Function [System] [page 613]
Returns the user name for the specified user ID.

UUIDTOSTR Function [String] [page 615]
Converts a unique identifier value (UUID, also known as GUID) to a string value.

VAR_POP Function [Aggregate] [page 616]
Computes the statistical variance of a population consisting of a numeric-expression, as a DOUBLE.

VAR_SAMP Function [Aggregate] [page 618]
Computes the statistical variance of a sample consisting of a numeric-expression, as a DOUBLE.

VAREXISTS Function [Miscellaneous] [page 620]
Returns 1 if a user-defined variable exists with the specified name. Returns 0 if no such variable exists.

VARIANCE Function [Aggregate] [page 622]
An alias for VAR_SAMP.

WATCOMSQL Function [Miscellaneous] [page 622]
Rewrites a Transact-SQL statement in the Watcom SQL dialect. This can be useful when converting existing Adaptive Server Enterprise stored procedures into Watcom SQL syntax.

WEEKS Function [Date and Time] [page 623]
Manipulates a TIMESTAMP or returns the number of weeks between two TIMESTAMP values.

WRITE_CLIENT_FILE Function [String] [page 625]
Creates and writes to a file on the client computer.

XMLAGG Function [Aggregate] [page 626]
Generates a forest of XML elements from a collection of XML values.

XMLCONCAT Function [String] [page 628]
Produces a forest of XML elements.

XMLELEMENT Function [String] [page 629]
Produces an XML element within a query.

XMLFOREST Function [String] [page 632]
Generates a forest of XML elements.

XMLGEN Function [String] [page 634]
Generates an XML value based on an XQuery constructor.

YEAR Function [Date and Time] [page 635]
Returns the year component of the TIMESTAMP argument.

YEARS Function [Date and Time] [page 636]
Manipulates a TIMESTAMP or returns the number of years between two TIMESTAMP values.

YMD Function [Date and Time] [page 638]
Returns a date value corresponding to the given year, month, and day of the month. Arguments are INTEGER values from -32768 to 32767.

Related Information

Function Types [page 207]

1.3.2.1 ABS Function [Numeric]

Returns the absolute value of a numeric expression.

Syntax

```
ABS( numeric-expression )
```

Parameters

numeric-expression

The number whose absolute value is to be returned.

Returns

An absolute value of the numeric expression.

Numeric-expression data type	Returns
INT	INT
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMERIC	NUMERIC

Standards

ANSI/ISO SQL Standard

Part of optional Language Feature T441.

Example

The following statement returns the value 66:

```
SELECT ABS( -66 );
```

1.3.2.2 ACOS Function [Numeric]

Returns the arc-cosine, in radians, of a numeric expression.

Syntax

```
ACOS( numeric-expression )
```

Parameters

numeric-expression

The cosine of the angle.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the arc-cosine value for 0.52:

```
SELECT ACOS( 0.52 );
```

Related Information

ASIN Function [Numeric] [page 251]
ATAN Function [Numeric] [page 252]
ATAN2 Function [Numeric] [page 253]
COS Function [Numeric] [page 298]

1.3.2.3 ARGN Function [Miscellaneous]

Returns a selected argument from a list of arguments.

Syntax

```
ARGN( integer-expression , expression [ , ... ] )
```

Parameters

integer-expression

The position of an argument within the list of expressions.

expression

An expression of any data type passed into the function. All supplied expressions must be of the same data type.

Returns

Using the value of the `integer-expression` as n, returns the nth argument (starting at 1) from the remaining list of arguments.

Remarks

While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 6:

```
SELECT ARGN( 6, 1,2,3,4,5,6 );
```

1.3.2.4 ARRAY Constructor [Composite]

Returns elements of a specific data type.

Syntax

```
ARRAY( [ expression [ , expression ... ] | single-column-query-expression ] )
```

Parameters

expression

An element expression in the ROW type.

single-column-query-expression

A query statement that returns a single column.

Returns

Array value

Remarks

All expressions must be union compatible.

An ARRAY type can contain other ARRAY or ROW values or be part of a ROW type.

The FETCH statement supports the transfer of values into an array. You can fetch values into an array for individual expressions, for entire arrays, or for portions of arrays.

Specific values or vectors of values can be dereferenced by using double square brackets.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Oracle

VARRAY can be used as a synonym of ARRAY.

Example

The following example illustrates how to construct an array:

```
SELECT FIRST f[[2]] FROM ( SELECT ARRAY( ID, Quantity ) FROM GROUPO.Products )
  AS dt( f ) ORDER BY f[[1]] ASC;
```

In this example, for each row of the Products table the ARRAY constructor builds an ARRAY type with two elements: the column ID and the value of the Quantity column, both of which are integers. The result is ordered by the first element of the array in each row, and the result returned is the second element from the array with the smallest first element (the Product ID of 300). You can also construct an array directly from a single-column query expression.

The following example illustrates an alternative way of constructing an array:

```
SELECT * FROM GROUPO.SalesOrders S WHERE ARRAY( SELECT P.ID FROM
GROUPO.Products P  JOIN GROUPO.SalesOrderItems SI ON( P.ID = SI.ProductID )AND
SI.ID = S.ID
ORDER  BY P.ID )< ARRAY ( SELECT ID FROM GROUPO.Products ORDER BY ID );
```

In the following example, the query's SELECT list uses three arrays: one produces a GROUP BY expression, and the MAX function uses the others. Each ARRAY type is de-referenced for a specific element before the result is returned to the client:

```
SELECT FIRST ARRAY( Quantity )[[1]], MAX( ARRAY( ID,Quantity ) )[[1]],
      MAX( ARRAY( name,name ))[[2]] FROM Products GROUP BY ARRAY( Quantity )
      ORDER BY 1;
```

The following example illustrates how to use the FETCH statement to transfer values into an array:

```
BEGIN
DECLARE product_orders ARRAY(10) OF ARRAY OF INTEGER;
DECLARE products ARRAY(10) OF INTEGER;
DECLARE greatest_orders INTEGER = 0;
DECLARE i INTEGER = 1;
DECLARE curs CURSOR FOR
  SELECT ProductID,
  ARRAY_AGG( Quantity ) AS Quantities
```

```
    FROM GROUPO.SalesOrderItems
    GROUP BY ProductID
    ORDER BY ProductID;
OPEN curs;
lp: LOOP
  FETCH NEXT curs INTO products[[i]], product_orders[[i]];
  IF SQLCODE <> 0 THEN LEAVE lp; END IF;
  IF i = 1 THEN
    SET greatest_orders = 1;
  ELSE
    IF CARDINALITY( product_orders[[greatest_orders]] )
        < CARDINALITY( product_orders[[i]] ) THEN
      SET greatest_orders = i;
    END IF;
  END IF;
  SET i = i + 1;
END LOOP;
IF greatest_orders >= 1 THEN
  SELECT * FROM GROUPO.Products WHERE ID = products[[greatest_orders]];
END IF;
END;
```

The following example uses the ARRAY constructor to initialize an array to a list of constants. The second element of the day-of-the-week array is selected.

```
BEGIN
   DECLARE @dow ARRAY( 7 ) OF CHAR(3) = ARRAY( 'Sun', 'Mon', 'Tue', 'Wed',
'Thu', 'Fri', 'Sat' );
   SELECT @dow[[2]];
END;
```

The following example uses the ARRAY and ROW constructors to initialize an array of rows. Each row consists of the name and age of a person.

```
BEGIN
    DECLARE people ARRAY( 3 ) OF ROW( name VARCHAR(64), age INT ) =
        ARRAY( ROW( 'Jack', 49 ), ROW( 'Dany', 22 ), ROW( 'Lara', 31 ) );
    SELECT people[[row_num]].name AS name, people[[row_num]].age AS age
        FROM sa_rowgenerator ( 1, CARDINALITY(people) ) ORDER BY row_num;
END;
```

The following example uses the ARRAY and ROW constructors to initialize an array of rows. Each row's members are initially NULL. The array elements are filled in using SET statements.

```
BEGIN
    DECLARE people ARRAY( 3 ) OF ROW( name VARCHAR(64), age INT ) =
        ARRAY( ROW(), ROW(), ROW() );
    SET people[[1]].name = 'Jack'; SET people[[1]].age = 49;
    SET people[[2]].name = 'Dany'; SET people[[2]].age = 22;
    SET people[[3]].name = 'Lara'; SET people[[3]].age = 31;
    SELECT people[[row_num]].name AS name, people[[row_num]].age AS age
        FROM sa_rowgenerator ( 1, CARDINALITY(people) ) ORDER BY row_num;
END;
```

The following example uses the ARRAY and ROW constructors. Each array element is initialized using a SET statement with a ROW constructor.

```
BEGIN
    DECLARE people ARRAY( 3 ) OF ROW(name VARCHAR(64), age INT);
    SET people[[1]] = ROW('Jack',49);
    SET people[[2]] = ROW('Dany',22);
    SET people[[3]] = ROW('Lara',31);
    SELECT people[[row_num]].name,people[[row_num]].age
```

```
        FROM sa_rowgenerator ( 1, 3 ) ORDER BY row_num;
END;
```

The following example uses the ARRAY and ROW constructors to initialize an ARRAY variable using a SET statement.

```
CREATE OR REPLACE VARIABLE arrayvar ARRAY OF ROW(
    a VARCHAR(32),
    b ARRAY OF ROW( b1 LONG NVARCHAR, b2 LONG NVARCHAR),
    c BIT,
    d NUMERIC(5,2)
);
SET arrayvar = ARRAY( ROW( 'first', ARRAY( ROW( 'Hello', 'John' ),
ROW( 'Goodbye', 'Mary' ) ), 1, 12.34 ),
                      ROW( 'second', ARRAY( ROW( 'Bonjour', 'Jean' ), ROW( 'Au
revoir', 'Marie' ) ), 0, 56.78 ) );
SELECT arrayvar[[x.row_num]].a AS a,
    arrayvar[[x.row_num]].b[[y.row_num]].b1 AS b1,
    arrayvar[[x.row_num]].b[[y.row_num]].b2 AS b2,
    arrayvar[[x.row_num]].c AS c,
    arrayvar[[x.row_num]].d AS d
    FROM sa_rowgenerator(1,CARDINALITY(arrayvar)) AS x,
         sa_rowgenerator(1,CARDINALITY(arrayvar[[1]].b)) AS y;
```

The following examples show ARRAY of ROW:

```
BEGIN
    DECLARE people ARRAY( 3 ) OF ROW(name VARCHAR(64), age INT) =
ARRAY( ROW('Jack',49), ROW('Dany',22), ROW('Lara',31) );
    SELECT people[[row_num]].name,people[[row_num]].age FROM sa_rowgenerator
( 1, 3 ) ORDER BY row_num;
END;
BEGIN
    DECLARE people ARRAY( 3 ) OF ROW(name VARCHAR(64), age INT) =
ARRAY( ROW(NULL,NULL), ROW(NULL,NULL), ROW(NULL,NULL) );
    SET people[[1]].name = 'Jack'; SET people[[1]].age = 49;
    SET people[[2]].name = 'Dany'; SET people[[2]].age = 22;
    SET people[[3]].name = 'Lara'; SET people[[3]].age = 31;
    SELECT people[[row_num]].name,people[[row_num]].age FROM sa_rowgenerator
( 1, 3 ) ORDER BY row_num;
END;
BEGIN
    DECLARE people ARRAY( 3 ) OF ROW(name VARCHAR(64), age INT);
    SET people[[1]] = ROW('Jack',49);
    SET people[[2]] = ROW('Dany',22);
    SET people[[3]] = ROW('Lara',31);
    SELECT people[[row_num]].name,people[[row_num]].age FROM sa_rowgenerator
( 1, 3 ) ORDER BY row_num;
END;
```

Related Information

ROW and ARRAY Composite Data Types [page 185]
Composite Functions [page 210]
Comparisons of Composite Types [page 198]
FETCH Statement [ESQL] [SP] [page 1162]

1.3.2.5 ARRAY_AGG Function [Aggregate]

Creates an unbounded, single-dimensional array from the specified expression for each group where the array element type is identical to the specified expression.

Syntax

```
ARRAY_AGG( expression
[ ORDER BY order-by-expression [ ASC | DESC ] , ... ] )
```

Parameters

expression

The expression to base the array on. The array is created with the first element having the value of the first group from `expression`, the second element having the value of the second group, and so on.

order-by-expression

Determines the order of the rows returned by `expression`. If `order-by-expression` is not specified, the order of the returned rows is not deterministic.

order-by-expression

Order the items returned by the function. There is no comma preceding this argument, which makes it easy to use in the case where no `delimiter-string` is supplied.

`order-by-expression` cannot be an integer literal. However, it can be a variable that contains an integer literal.

When an ORDER BY clause contains constants, they are interpreted by the optimizer and then replaced by an equivalent ORDER BY clause. For example, the optimizer interprets ORDER BY 'a' as ORDER BY expression.

A query block containing more than one aggregate function with valid ORDER BY clauses can be executed if the ORDER BY clauses can be logically combined into a single ORDER BY clause. For example, the following clauses:

```
ORDER BY expression1, 'a', expression2
```

```
ORDER BY expression1, 'b', expression2, 'c', expression3
```

are subsumed by the clause:

```
ORDER BY expression1, expression2, expression3
```

Returns

ARRAY

Remarks

Array elements are filled from the input beginning with the first element.

ARRAY_AGG does not ignore NULL values in its input. NULL values are stored in the array as separate elements like any other value. If the group is empty, the result of the ARRAY_AGG function contains a NULL element for that group.

ARRAY_AGG cannot be used as a window function, but it can be used as an input to a window function.

The UNNEST array operator can be used to create a series of rows from an array to process each array element with other relational expressions.

Standards

ANSI/ISO SQL Standard

Feature S098.

Example

The following statements illustrate how to generate an array that contains the list of all product colors. This array could then be passed as a parameter to a procedure that checks whether there were any non-standard colors in the table, for example.

```
CREATE VARIABLE color_list ARRAY OF LONG VARCHAR;
SELECT ARRAY_AGG(DISTINCT Color) INTO color_list FROM Products;
```

Related Information

UNNEST Array Operator [page 28]
LIST Function [Aggregate] [page 439]

1.3.2.6 ARRAY_MAX_CARDINALITY Function [Composite]

Returns the maximal number of elements in the array.

Syntax

```
ARRAY_MAX_CARDINALITY( array-expression )
```

Parameters

array-expression

The array expression to evaluate.

If `array-expression` is NULL, then ARRAY_MAX_CARDINALITY returns NULL.

Returns

INTEGER

Remarks

The cardinality of the collection is the number of elements in the collection.

For an unbounded array, ARRAY_MAX_CARDINALITY returns the maximum size limit of a supported array. For a bounded array or an array composed via a constructor, ARRAY_MAX_CARDINALITY returns the maximum explicitly or implicitly declared size of the array.

Standards

ANSI/ISO SQL Standard

Feature S403.

Related Information

ARRAY_AGG Function [Aggregate] [page 247]
CARDINALITY Function [Composite] [page 274]
TRIM_ARRAY Function [Composite] [page 603]

1.3.2.7 ASCII Function [String]

Returns the integer ASCII value of the first byte in a string-expression.

Syntax

```
ASCII( string-expression )
```

Parameters

string-expression

The string.

Returns

SMALLINT

Remarks

If the string is empty, then ASCII returns zero. Literal strings must be enclosed in quotes. If the database character set is multibyte and the first character of the parameter string consists of more than one byte, the result is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 90:

```
SELECT ASCII( 'Z' );
```

Related Information

String Functions [page 222]
CHAR Function [String] [page 279]

1.3.2.8 ASIN Function [Numeric]

Returns the arc-sine, in radians, of a number.

Syntax

```
ASIN( numeric-expression )
```

Parameters

numeric-expression

The sine of the angle.

Returns

DOUBLE

Remarks

The SIN and ASIN functions are inverse operations.

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the arc-sine value for 0.52:

```
SELECT ASIN( 0.52 );
```

Related Information

ACOS Function [Numeric] [page 241]
ATAN Function [Numeric] [page 252]
ATAN2 Function [Numeric] [page 253]
SIN Function [Numeric] [page 553]

1.3.2.9 ATAN Function [Numeric]

Returns the arc-tangent, in radians, of a number.

Syntax

```
ATAN( numeric-expression )
```

Parameters

numeric-expression

The tangent of the angle.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

The ATAN and TAN functions are inverse operations.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the arc-tangent value for 0.52:

```
SELECT ATAN( 0.52 );
```

Related Information

ACOS Function [Numeric] [page 241]
ASIN Function [Numeric] [page 251]
ATAN2 Function [Numeric] [page 253]
TAN Function [Numeric] [page 587]

1.3.2.10 ATAN2 Function [Numeric]

Returns the arc-tangent, in radians, of the ratio of two numbers.

Syntax

```
{ ATAN2 | ATN2 } ( numeric-expression-1 , numeric-expression-2 )
```

Parameters

numeric-expression-1

The numerator in the ratio whose arc-tangent is calculated.

numeric-expression-2

The denominator in the ratio whose arc-tangent is calculated.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

UltraLite does not support the function name short form *ATN2*.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the arc-tangent value for the ratio 0.52 to 0.60:

```
SELECT ATAN2( 0.52, 0.60 );
```

Related Information

ACOS Function [Numeric] [page 241]
ASIN Function [Numeric] [page 251]
ATAN Function [Numeric] [page 252]
TAN Function [Numeric] [page 587]

1.3.2.11 AVG Function [Aggregate]

Computes the average, for a set of rows, of a numeric expression or of a set of unique values.

Syntax

Numeric expressions

```
AVG( [ ALL | DISTINCT ] numeric-expression )
```

Window function

```
AVG( [ ALL ] numeric-expression) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

UltraLite - numeric expressions

```
AVG( [ DISTINCT ] numeric-expression )
```

Parameters

[ALL] numeric-expression

The expression whose average is calculated over the rows in each group.

DISTINCT clause

Computes the average of the unique numeric values in each group.

Returns

Returns the NULL value for a group containing no rows.

Returns DOUBLE if the argument is DOUBLE, otherwise NUMERIC.

Remarks

This average does not include rows where the `numeric-expression` is the NULL value.

This function can generate an overflow error, resulting in an error being returned. You can use the CAST function on `numeric-expression` to avoid the overflow error.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

Core Feature. The `numeric-expression` syntax is a Core Feature of the Standard, while `window-spec` syntax comprises part of optional Language Feature T611, "Basic OLAP operations". The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional Language feature F561, "Full value expressions". The software also supports Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references. The software does not support optional Language Feature F442, "Mixed column references in set functions", and it also does

not permit the arguments of an aggregate function to include both a column reference from the query block containing the AVG function, combined with an outer reference.

Example

The following statement returns the value 49988.623200 when connected to the SQL Anywhere 17 Demo:

```
SELECT AVG( Salary ) FROM Employees;
```

The following statement returns the average product price from the Products table when connected to the SQL Anywhere 17 Demo database:

```
SELECT AVG( DISTINCT UnitPrice ) FROM Products;
```

The following statement returns an error with SQLSTATE 42W68 because the arguments of AVG contain both a quantified expression from the subquery, and an outer reference (p.Quantity) from the outer SELECT block when connected to the SQL Anywhere 17 Demo:

```
SELECT * from GROUPO.Products as p
WHERE p.Quantity > ( SELECT AVG( 0.5 * p.Quantity + 0.5 * s.Quantity )
                     from GROUPO.SalesOrderItems as s
                     WHERE s.ProductID = p.ProductID )
```

Related Information

CAST Function [Data Type Conversion] [page 275]
SUM Function [Aggregate] [page 581]
COUNT Function [Aggregate] [page 300]
Troubleshooting Database Upgrades: Aggregate Functions and Outer References

1.3.2.12 BASE64_DECODE Function [String]

Decodes data using the MIME base64 format and returns the string as a LONG VARCHAR.

Syntax

```
BASE64_DECODE( string-expression )
```

Parameters

string-expression

The string that is to be decoded. The string must be base64-encoded.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following inserts an image into an image table from an Embedded SQL program. The input data (host variable) must be base64 encoded:

```
EXEC SQL INSERT INTO images ( image_data ) VALUES ( BASE64_DECODE ( :img ) );
```

Related Information

String Functions [page 222]
BASE64_ENCODE Function [String] [page 257]

1.3.2.13 BASE64_ENCODE Function [String]

Encodes data using the MIME base64 format and returns it as a 7-bit ASCII string.

Syntax

```
BASE64_ENCODE( data-value )
```

Parameters

data-value

The string that is to be encoded.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following retrieves data from a fictitious table containing images and returns it in ASCII format. The resulting string can be embedded into an email message, and then decoded by the recipient to retrieve the original image.

```
SELECT BASE64_ENCODE( image_data ) FROM IMAGES;
```

Related Information

String Functions [page 222]
BASE64_DECODE Function [String] [page 256]

1.3.2.14 BINTOHEX Function [Data Type Conversion]

Returns the hexadecimal equivalent of a binary string.

Syntax

```
BINTOHEX( binary-expression )
```

Parameters

binary-expression

The binary string to be converted to a hexadecimal string.

Returns

The BINTOHEX function returns a LONG VARCHAR string. The length of the result is twice the length of the input string.

Remarks

The CAST, CONVERT, BINTOHEX, HEXTOBIN, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns a string containing 313233:

```
SELECT BINTOHEX(0x313233);
```

Related Information

Converting to and from Hexadecimal Values [page 12]
HEXTOBIN Function [Data Type Conversion] [page 399]

1.3.2.15 BIT_AND Function [Aggregate]

Returns the bit-wise AND of the specified expression for each group of rows.

Syntax

```
BIT_AND( bit-expression )
```

Parameters

bit-expression

The object to be aggregated. The expression can be a VARBIT array, a BINARY value, or an INTEGER (including all integer variants such as BIT and TINYINT).

Returns

The same data type as the argument. For each bit position compared, if every row has a 1 in the bit position, return 1; otherwise, return 0.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example generates four rows containing a CHAR column, then converts the values to VARBIT:

```
SELECT BIT_AND( CAST( row_value AS VARBIT ) )
FROM dbo.sa_split_list( '0001,0111,0100,0011' );
```

The result 0000 is determined as follows:

1. A bitwise AND is performed between row 1 (0001) and row 2 (0111), resulting in 0001 (both values had a 1 in the fourth bit).
2. A bitwise AND is performed between the result from the previous comparison (0001) and row 3 (0100), resulting in 0000 (neither value had a 1 in the same bit).
3. A bitwise AND is performed between the result from the previous comparison (0000) and row 4 (0011), resulting in 0000 (neither value had a 1 in the same bit).

Related Information

BIT_OR Function [Aggregate] [page 262]
BIT_XOR Function [Aggregate] [page 265]
Bitwise Operators [page 33]

1.3.2.16 BIT_LENGTH Function [Bit Array]

Returns the number of bits stored in the array.

Syntax

```
BIT_LENGTH( bit-expression )
```

Parameters

bit-expression

The bit expression for which the length is to be determined.

Returns

INT

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 8:

```
SELECT BIT_LENGTH( '01101011' );
```

Related Information

CHAR_LENGTH Function [String] [page 280]

1.3.2.17 BIT_OR Function [Aggregate]

Returns the bit-wise OR of the specified expression for each group of rows.

Syntax

```
BIT_OR( bit-expression )
```

Parameters

bit-expression

The object to be aggregated. The expression can be a VARBIT array, a BINARY value, or an INTEGER (including all integer variants such as BIT and TINYINT).

Returns

The same data type as the argument. For each bit position compared, if any row has a 1 in the bit position, this function returns 1; otherwise, it returns 0.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example generates four rows containing a CHAR column, then converts the values to VARBIT:

```
SELECT BIT_OR( CAST( row_value AS VARBIT ) )
FROM dbo.sa_split_list( '0001,0111,0100,0011' );
```

The result 0111 is determined as follows:

1. A bitwise OR is performed between row 1 (0001) and row 2 (0111), resulting in 0111.
2. A bitwise OR is performed between the result from the previous comparison (0111) and row 3 (0100), resulting in 0111.
3. A bitwise OR is performed between the result from the previous comparison (0111) and row 4 (0011), resulting in 0111.

Related Information

BIT_AND Function [Aggregate] [page 260]
BIT_XOR Function [Aggregate] [page 265]
Bitwise Operators [page 33]

1.3.2.18 BIT_SUBSTR Function [Bit Array]

Returns a sub-array of a bit array.

Syntax

```
BIT_SUBSTR( bit-expression , start [ , length ] )
```

Parameters

bit-expression

The bit array from which the sub-array is to be extracted.

start

The start position of the sub-array to return. A negative starting position specifies the number of bits from the end of the array instead of the beginning. The first bit in the array is at position 1.

length

The length of the sub-array to return. A positive length specifies that the sub-array ends `length` bits to the right of the starting position, while a negative length returns, at most, `length` bits up to, and including, the starting position, from the left of the starting position.

Returns

LONG VARBIT

Remarks

Both `start` and `length` can be either positive or negative. Using appropriate combinations of negative and positive numbers, you can get a sub-array from either the beginning or end of the string. Using a negative number for `length` does not impact the order of the bits returned in the sub-array.

If `length` is specified, the sub-array is restricted to that length. If `start` is zero and `length` is non-negative, a start value of 1 is used. If `start` is zero and `length` is negative, a start value of -1 is used.

If `length` is not specified, selection continues to the end of the array.

The BIT_SUBSTR function is equivalent to, but faster than, the following:

```
CAST( SUBSTR( CAST( bit-expression AS VARCHAR ),
start [, length ] )
AS VARBIT );
```

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns 1101:

```
SELECT BIT_SUBSTR( '001101', 3 );
```

The following statement returns 10110:

```
SELECT BIT_SUBSTR( '01011011101111011111', 2, 5 );
```

The following statement returns 11111:

```
SELECT BIT_SUBSTR( '01011011101111011111', -5, 5 );
```

Related Information

SUBSTRING Function [String] [page 578]

1.3.2.19 BIT_XOR Function [Aggregate]

Returns the bit-wise XOR of the specified expression for each group of rows.

Syntax

```
BIT_XOR( bit-expression )
```

Parameters

bit-expression

The object to be aggregated. The expression can be a VARBIT array, a BINARY value, or an INTEGER (including all integer variants such as BIT and TINYINT).

Returns

The same data type as the argument. For each bit position compared, if an odd number of rows have a 1 in the bit position, return 1; otherwise, return 0.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example generates four rows containing a CHAR column, then converts the values to VARBIT:

```
SELECT BIT_XOR( CAST( row_value AS VARBIT ) )
FROM dbo.sa_split_list( '0001,0111,0100,0011' );
```

The result `0001` is determined as follows:

1. A bitwise exclusive OR (XOR) is performed between row 1 (0001) and row 2 (0111), resulting in 0110.
2. A bitwise XOR is performed between the result from the previous comparison (0110) and row 3 (0100), resulting in 0010.
3. A bitwise XOR is performed between the result from the previous comparison (0010) and row 4 (0011), resulting in 0001.

Related Information

BIT_AND Function [Aggregate] [page 260]
BIT_OR Function [Aggregate] [page 262]
Bitwise Operators [page 33]

1.3.2.20 BYTE_INSERTSTR Function [String]

Inserts a string into another string at a position specified in bytes.

Syntax

```
BYTE_INSERTSTR( insert-position , source-string , insert-string )
```

Parameters

insert-position

The byte position after which `insert-string` is to be inserted. The first byte in the string is position 1. Use 0 to insert before the first byte in the string.

source-string

The string into which `insert-string` is to be inserted. `source-string` can be any length.

insert-string

The string to be inserted.

Returns

LONG BINARY

Remarks

The arguments `source-string` and `insert-string` are treated as binary strings.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement inserts the string 123456 after the 5th byte position, and returns the value `0xfedcba9876123456543210`:

```
SELECT BYTE_INSERTSTR(5,0xfedcba9876543210,0x123456);
```

1.3.2.21 BYTE_LENGTH Function [String]

Returns the number of bytes in a string.

Syntax

```
BYTE_LENGTH( string-expression )
```

Parameters

string-expression

The string whose length is to be calculated.

Returns

INT

Remarks

Trailing white space characters in the `string-expression` are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the BYTE_LENGTH value may differ from the number of characters returned by CHAR_LENGTH.

This function supports NCHAR inputs and/or outputs.

UltraLite: UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard. The equivalent function is the OCTET_LENGTH function.

Example

The following statement returns the value 12:

```
SELECT BYTE_LENGTH( 'Test Message' );
```

Related Information

String Functions [page 222]
CHAR_LENGTH Function [String] [page 280]
DATALENGTH Function [System] [page 313]
LENGTH Function [String] [page 436]

1.3.2.22 BYTE_LOCATE Function [String]

Returns the position of one BYTE string within another.

Syntax

```
BYTE_LOCATE( source-string , search-string [ , start-position ] )
```

Parameters

source-string

The string to be searched.

search-string

The string to be searched for.

start-position

The byte position in the string to begin the search. The first byte is position 1. If the starting offset is negative, then the BYTE_LOCATE function returns the last matching string offset rather than the first, as counted from the end of the string. A negative offset indicates how much of the end of the string is to be excluded from the search. The number of bytes excluded is calculated as (-1 * offset) -1.

Although `start-position` acts as an offset for where the search is started, the return value still reflects the actual starting position of the matching string, regardless of where the search was started.

Returns

INTEGER

Remarks

If `start-position` is specified, then the search starts at that offset into the string.

`source-string` can be any length, but `search-string` is limited to 255 bytes. If `search-string` is longer than 255 bytes, then the function returns a NULL value. If `search-string` is not found, then 0 is returned. Searching for a zero-length `search-string` returns 1. If any of the arguments are NULL, then the result is NULL.

`source-string` and `search-string` can be any data type that can be converted to a binary data type; binary comparisons are used.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

When `start-position` is any positive number from 0 to 8 inclusive, the following statement returns 8, indicating that the first matching byte in the string is at position 8. If `start-position` is greater than 8, then the example returns 0 because the string 'party' is not found after position 8.

```
SELECT BYTE_LOCATE(
   'office party this week - rsvp as soon as possible',
   'party',
   2 );
```

When `start-position` is any number from 0 to -38 inclusive, the following statement returns 8. Any number lower than -38 returns 0. This indicates that the first matching byte in the string is at position 8, and indicates that the position of the last matching byte (the 'y' in 'party') is in position 38 as counted backwards from the end of the string.

```
SELECT BYTE_LOCATE(
   'office party this week - rsvp as soon as possible',
```

```
    'party',
    -38 );
```

1.3.2.23 BYTE_REPLACE Function [String]

Replaces a string with another string, and returns the new result.

Syntax

```
BYTE_REPLACE( source-string , search-string , replace-string )
```

Parameters

source-string

The string to be searched.

search-string

The string to be searched for and replaced by `replace-string`. `search-string` is limited to 255 bytes. If `search-string` is an empty string, then `source-string` is returned unchanged.

replace-string

The string that replaces all instances of `search-string`. If `replacement-string` is an empty string, then all occurrences of `search-string` are deleted.

Returns

LONG BINARY

Remarks

All instances of `search-string` are replaced with `replace-string`.

`source-string`, `search-string`, and `replace-string` can be any data type that can be converted to a binary data type; binary comparisons are used.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the binary value `0x78782e6465662e78782e676869`, which is the hexadecimal representation of the string `xx.def.xx.ghi`:

```
SELECT BYTE_REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' );
```

1.3.2.24 BYTE_STUFF Function [String]

Deletes multiple bytes from one string and replaces them with different bytes.

Syntax

```
BYTE_STUFF( source-string , start-position , length , insert-string )
```

Parameters

source-string

The byte string to be modified by the BYTE_STUFF function. `source-string` can be any length.

start-position

The byte position at which to begin deleting bytes. The first byte in the string is position 1.

length

The number of bytes to delete.

insert-string

The string to be inserted. To delete a portion of a string using the BYTE_STUFF function, use a NULL replacement string.

Returns

LONG BINARY

Remarks

The arguments `source-string` and `insert-string` can be any data type that can be converted to a binary data type, and are treated as binary strings.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement inserts the hexadecimal string 123456 starting at the 6th byte position, and returns the binary value `0xfedcba9876123456543210`:

```
SELECT BYTE_STUFF(0xfedcba9876543210, 6, 0, 0x123456);
```

1.3.2.25 BYTE_SUBSTR Function [String]

Returns a substring of a string. The substring is determined using bytes, not characters.

Syntax

```
BYTE_SUBSTR( source-string , start-position [ , length ] )
```

Parameters

source-string

The data from which the binary substring is taken.

start-position

An integer expression indicating the start of the substring. A positive integer starts from the beginning of the data, with the first byte being position 1. A negative integer specifies a substring starting from the end of the data, the final byte being at position -1.

length

An integer expression indicating the length of the substring. A positive `length` specifies the number of bytes to be taken *starting* at the start position. A negative `length` returns at most `length` bytes up to, and including, the starting position, from the left of the starting position.

Returns

BINARY or LONG BINARY, depending on the length of the result.

Remarks

Both `start-position` and `length` can be either positive or negative. Use appropriate combinations of negative and positive numbers, to get a substring from either the beginning or end of the string. If `length` is specified, the maximum length of the substring is ABS(`length`).

If `start-position` is zero and length is non-negative, then a `start-position` value of 1 is used. If `start-position` is zero and `length` is negative, then a start value of -1 is used.

The argument `source-string` can be any data type that can be converted to a binary data type, and is treated as a binary string.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the binary value `0x54657374`, which is the hexadecimal representation of `Test`:

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

Related Information

String Functions [page 222]
SUBSTRING Function [String] [page 578]

1.3.2.26 CARDINALITY Function [Composite]

Returns the highest number of any array element that has been assigned a value, including NULL.

Syntax

```
CARDINALITY( array-expression )
```

Parameters

array-expression

The array expression on which the cardinality is calculated.

If `array-expression` is NULL, then CARDINALITY returns NULL.

Returns

INTEGER

Remarks

The cardinality of the collection is the number of elements in the collection.

The result is an integer between zero and the maximum size of the array.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the value 4:

```
SELECT CARDINALITY( ARRAY( 3,4 ) || ARRAY( 5,6 ) );
```

Related Information

ARRAY_AGG Function [Aggregate] [page 247]
ARRAY_MAX_CARDINALITY Function [Composite] [page 248]
TRIM_ARRAY Function [Composite] [page 603]

1.3.2.27 CAST Function [Data Type Conversion]

Returns the value of an expression converted to a supplied data type.

Syntax

```
CAST( expression AS datatype )
```

Parameters

expression

The expression to be converted.

datatype

The data type to cast the expression into. Set the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in a table or view, or to the data type of a variable.

Returns

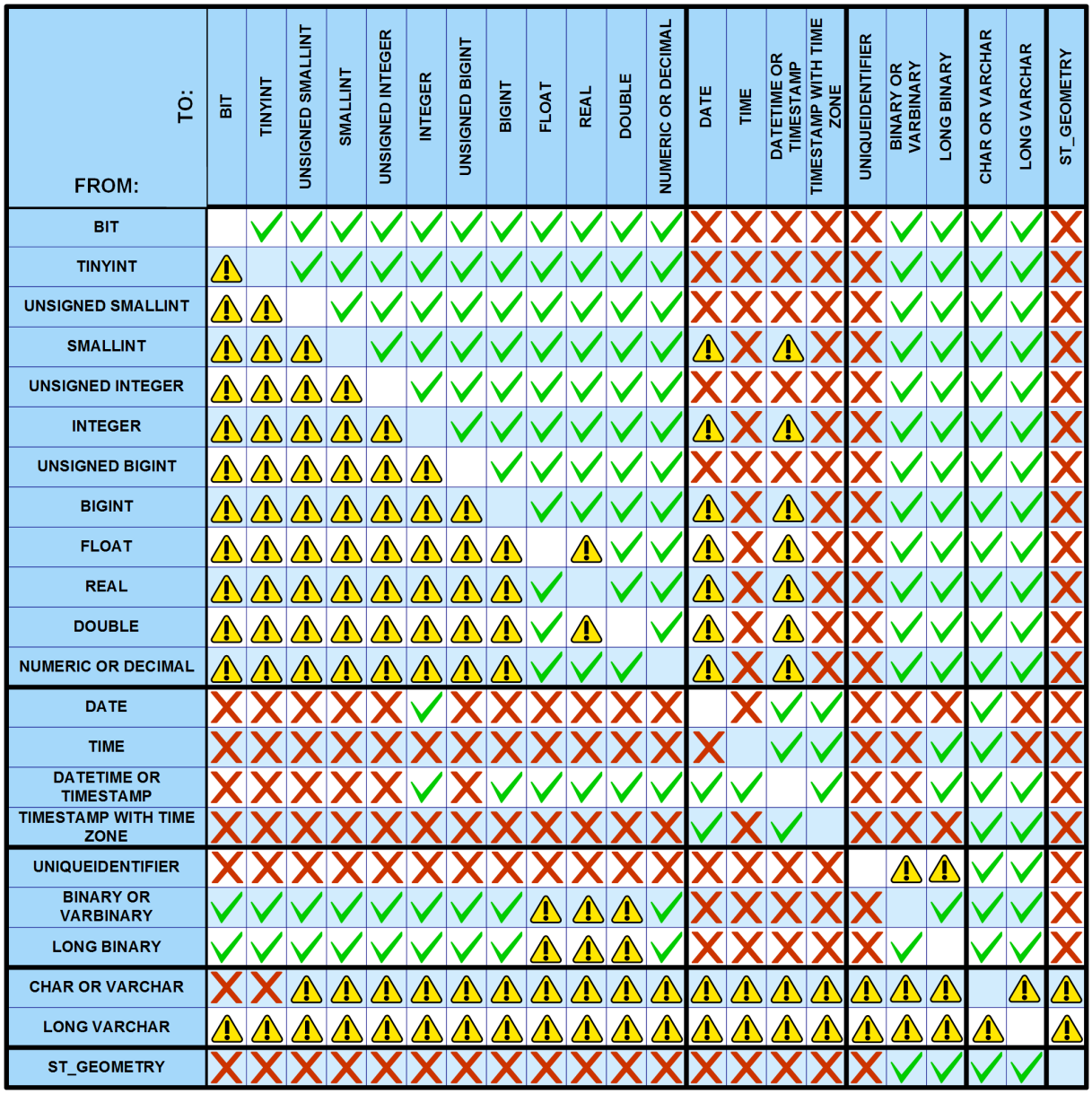
Depends on the data type requested.

Remarks

If you use the CAST function to truncate strings, then the string_rtruncation database option must be set to OFF; otherwise, there will be an error. Use the LEFT function to truncate strings.

If you do not indicate a length for character string types, then an appropriate length is chosen. If neither precision nor scale is specified for a DECIMAL conversion, then the database server selects appropriate values.

UltraLite: It is recommended that you explicitly indicate the precision and scale in your CAST function. The ability to convert depends on the value used in the conversion. The values in the original data type must be compatible with the new data type to avoid generating a conversion error. Use the following chart to determine whether a conversion is supported:



Symbol	Compatibility
✓	Always converts
✗	Never converts
⚠	Value-dependent

> **i** Note
>
> In UltraLite:
>
> - To convert between a VARBINARY and a UNIQUEIDENTIFIER, the VARBINARY value must have a 16 byte length.
> - To convert between a NUMERIC and a VARBINARY, the NUMERIC source must have a value that can also be cast as a BIGINT.
> - To convert from a VARCHAR to an ST_GEOMETRY, the VARCHAR source must represent a valid geometry in either WKT or EWKT format.
> - To convert from a VARBINARY to an ST_GEOMETRY, the VARBINARY source must represent a valid geometry in WKB format.
> - When casting from a WKB or WKT formatted source to an ST_GEOMETRY, an SRID of 0 is assigned to the ST_GEOMETRY value. When casting from an ST_GEOMETRY, VARCHAR values are formatted in EWKT and VARBINARY values are formatted in WKB.

The HEXTOINT and INTTOHEX functions can be used to convert to and from hexadecimal values.

Standards

ANSI/ISO SQL Standard

Core Feature. However, in the software, CAST supports a number of data type conversions that are not permitted by the ANSI/ISO SQL Standard. For example, you can CAST an integer value to a DATE type, whereas in the ANSI/ISO SQL Standard this type of conversion is not permitted.

Example

The following function ensures a string is used as a date:

```
SELECT CAST( '2000-10-31' AS DATE );
```

The value of the expression `1 + 2` is calculated, and the result is then cast into a single-character string.

```
SELECT CAST( 1 + 2 AS CHAR );
```

Casting between VARCHAR and ST_GEOMETRY is usually implicit. For example, the following statement adds values to ST_GEOMETRY columns using the ST_POINT function and a VARCHAR. Each value is implicitly cast to an ST_GEOMETRY data type consistent with the table columns, but results still appear as VARCHAR.

```
INSERT INTO T1 VALUES (2, ST_POINT(1,2,0), 'SRID=2163;Point(1 2)');
```

The following statement casts a value to the data type defined for the BirthDate column (DATE data type) of the Employees table:

```
SELECT CAST ( '1966-10-30' AS Employees.BirthDate%TYPE );
```

UltraLite: You can use the CAST function to shorten strings:

```
SELECT CAST ( 'Surname' AS CHAR(5) );
```

Related Information

Data Type Conversions [page 200]
Converting to and from Hexadecimal Values [page 12]
CONVERT Function [Data Type Conversion] [page 294]
LEFT Function [String] [page 435]

1.3.2.28 CEILING Function [Numeric]

Returns the first integer that is greater or equal to a given value. For positive numbers, this is known as rounding up.

Syntax

```
{ CEILING | CEIL } ( numeric-expression )
```

Parameters

numeric-expression

The number whose ceiling is to be calculated.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

Standards

ANSI/ISO SQL Standard

The CEILING function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 60:

```
SELECT CEILING( 59.84567 );
```

Related Information

FLOOR Function [Numeric] [page 387]

1.3.2.29 CHAR Function [String]

Returns the character with the ASCII value of a number.

Syntax

```
CHAR( integer-expression )
```

Parameters

integer-expression

The number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive.

Returns

VARCHAR

Remarks

The character returned corresponds to the supplied numeric expression in the current database character set, according to a binary sort order.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value Y:

```
SELECT CHAR( 89 );
```

Related Information

String Functions [page 222]

1.3.2.30 CHAR_LENGTH Function [String]

Returns the number of characters in a string.

Syntax

```
CHAR_LENGTH ( string-expression )
```

Parameters

string-expression

The string whose length is to be calculated.

Returns

INT

Remarks

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the value returned by the CHAR_LENGTH function may differ from the number of bytes returned by the BYTE_LENGTH function.

You can use the CHAR_LENGTH function and the LENGTH function interchangeably for CHAR, VARCHAR, LONG VARCHAR, and NCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types. This function supports NCHAR inputs and/or outputs.

UltraLite: You can use the CHAR_LENGTH function and the LENGTH function interchangeably for CHAR, VARCHAR and LONG VARCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types.

Standards

ANSI/ISO SQL Standard

CHAR_LENGTH is a Core Feature. Using CHAR_LENGTH over an expression of type NCHAR comprises part of optional ANSI/ISO SQL Language Feature F421.

Example

The following statement returns the value 8:

```
SELECT CHAR_LENGTH( 'Chemical' );
```

Related Information

String Functions [page 222]
BYTE_LENGTH Function [String] [page 267]

1.3.2.31 CHARINDEX Function [String]

Returns the position of one string in another.

Syntax

```
CHARINDEX( string-expression-1 , string-expression-2 )
```

Parameters

string-expression-1

The string for which you are searching. The value must be less than 256 bytes.

string-expression-2

The string to be searched.

Returns

INT

Remarks

The first character of `string-expression-1` is identified as 1. If the string being searched contains more than one instance of the other string, then the CHARINDEX function returns the position of the first instance.

If the string being searched does not contain the other string, then the CHARINDEX function returns 0.

If any of the arguments are NULL, the result is NULL.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns last and first names from the Surname and GivenName columns of the Employees table, but only when the last name begins with the letter K:

```
SELECT Surname, GivenName
FROM GROUPO.Employees
WHERE CHARINDEX( 'K', Surname ) = 1;
```

The following results are returned:

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

Related Information

String Functions [page 222]
SUBSTRING Function [String] [page 578]
REPLACE Function [String] [page 525]
LOCATE Function [String] [page 442]

1.3.2.32 COALESCE Function [Miscellaneous]

Returns the first non-NULL expression from a list. This function is identical to the ISNULL function.

Syntax

```
COALESCE( expression , expression [ , ...] )
```

Parameters

expression

Any expression.

At least two expressions must be passed into the function, and all expressions must be comparable.

Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the first non-NULL expression from the list. If the database server cannot find a common comparison type, then an error is returned.

Remarks

The result is NULL only if all the arguments are NULL.

The parameters can be of any scalar type, but not necessarily same type.

Standards

ANSI/ISO SQL Standard

Core Feature.

Example

The following statement returns the value 34:

```
SELECT COALESCE( NULL, 34, 13, 0 );
```

Related Information

ISNULL Function [Miscellaneous] [page 429]

1.3.2.33 COMPARE Function [String]

Allows you to compare two character strings based on alternate collation rules.

Syntax

```
COMPARE( string-expression-1 , string-expression-2
[, { collation-id | collation-name [ ( collation-tailoring-string ) ] } ]
)
```

Parameters

string-expression-1

The first string expression.

string-expression-2

The second string expression.

The string expression can only contain characters that are encoded in the database's character set.

collation-id

A variable or integer constant that specifies the sort order to use. You can only use a `collation-id` for built-in collations.

If you do not specify a collation name or ID, the default is Default Unicode multilingual.

collation-name

A string or a character variable that specifies the name of the collation to use. You can also specify char_collation or db_collation (for example, `COMPARE( 'abc', 'ABC', 'char_collation' );`) to use the database's CHAR collation. Similarly, you can specify nchar_collation to use the database's NCHAR collation.

collation-tailoring-string

Optionally, you can specify collation tailoring options (`collation-tailoring-string`) for additional control over the character comparison. These options take the form of keyword=value pairs in parentheses, following the collation name. For example, `'UCA(locale=es;case=LowerFirst;accent=respect)'`. The syntax for specifying these options is identical to the syntax defined for the COLLATION clause of the CREATE DATABASE statement.

> **i Note**
>
> All the collation tailoring options are supported when specifying the UCA collation. For all other collations, only case sensitivity tailoring option is supported.

Returns

An INTEGER, based on the collation rules that you choose:

Value	Meaning
1	`string-expression-1` is greater than `string-expression-2`
0	`string-expression-1` is equal to `string-expression-2`
-1	`string-expression-1` is less than `string-expression-2`

Remarks

The COMPARE function does not equate empty strings and strings containing only spaces, even if the database has blank-padding enabled. The COMPARE function uses the SORTKEY function to generate collation keys for comparison.

If either `string-expression-1` or `string-expression-2` is NULL, the result is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example performs three comparisons using the COMPARE function:

```
SELECT COMPARE( 'abc','ABC','UCA(case=LowerFirst)' ),
       COMPARE( 'abc','ABC','UCA(case=Ignore)' ),
       COMPARE( 'abc','ABC','UCA(case=UpperFirst)' );
```

The values returned are -1, 0, 1, indicating the result of each comparison. The first comparison results in -1, indicating that `string-expression-2` ('ABC') is less than `string-expression-1` ('abc'). This is because case sensitivity is set to LowerFirst in the first COMPARE statement.

Related Information

String Functions [page 222]
SORTKEY Function [String] [page 556]
Collation Tailoring Options

1.3.2.34 COMPRESS Function [String]

Compresses the string and returns a value of type LONG BINARY.

Syntax

```
COMPRESS( string-expression [ , compression-algorithm-alias ] )
```

Parameters

string-expression

The string to be compressed. Binary values can be passed to this function. This parameter is case sensitive, even in case-insensitive databases.

compression-algorithm-alias

Alias for the algorithm to use for compression. The supported values are zip and gzip (both are based on the same algorithm, but use different headers and trailers). Zip is a widely supported compression algorithm. Gzip is compatible with the gzip utility on UNIX and Linux, whereas the zip algorithm is not.

Decompression must be performed with the same algorithm.

Returns

LONG BINARY

Remarks

The value returned by the COMPRESS is not human-readable. If the value returned is longer than the original string, its maximum size will not be larger than a 0.1% increase over the original string + 12 bytes. You can decompress a compressed `string-expression` using the DECOMPRESS function.

If you are storing compressed values in a table, the column should be BINARY or LONG BINARY so that character set conversion is not performed on the data.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the length of the binary string created by compressing the string 'Hello World' using the gzip algorithm. This example can be useful when you want to determine whether a value has a shorter length when compressed.

```
SELECT LENGTH( COMPRESS( 'Hello world', 'gzip' ) );
```

Related Information

String Functions [page 222]
DECOMPRESS Function [String] [page 339]

1.3.2.35 CONFLICT Function [Miscellaneous]

Indicates if a column is a source of conflict for an UPDATE being performed against a consolidated database in a SQL Remote environment.

Syntax

```
CONFLICT( column-name )
```

Parameters

column-name

The name of the column being tested for conflicts.

Returns

Returns TRUE if the column appears in the VERIFY list of an UPDATE statement executed by the SQL Remote Message Agent and if the value provided in the VALUES list of that statement does not match the original value of the column in the row being updated. Otherwise, returns FALSE.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The CONFLICT function is intended for use in SQL Remote RESOLVE UPDATE triggers to avoid error messages. To illustrate the use of the CONFLICT function, consider the following table:

```
CREATE TABLE Admin (
```

```
  PKey bigint NOT NULL DEFAULT GLOBAL AUTOINCREMENT,
  TextCol CHAR(20) NULL, PRIMARY KEY ( PKey ) );
```

Assume that consolidated and remote databases both have the following row in the Admin table:

```
1, 'Initial'
```

Now, at the consolidated database, update the row as follows:

```
UPDATE Admin SET TextCol = 'Consolidated Update' WHERE PKey = 1;
```

At the remote database, update the row to a different value as follows:

```
UPDATE Admin SET TextCol = 'Remote Update' WHERE PKey = 1;
```

Next, run dbremote on the remote database. It generates a message file with the following statements in it, to be executed at the consolidated database:

```
UPDATE Admin SET TextCol='Remote Update'
VERIFY ( TextCol )
VALUES ( 'Initial' )
WHERE PKey=1;
```

When the SQL Remote Message Agent runs at the consolidated database and applies this UPDATE statement, SQL Anywhere uses the VERIFY and VALUES clause to determine whether a RESOLVE UPDATE trigger will fire. A RESOLVE UPDATE trigger fires only when the update is executed from the SQL Remote Message Agent against a consolidated database. Here is a RESOLVE UPDATE trigger:

```
CREATE TRIGGER ResolveUpdateAdmin
RESOLVE UPDATE ON Admin
REFERENCING OLD AS OldConsolidated
    NEW AS NewRemote
    REMOTE as OldRemote
FOR EACH ROW BEGIN
  MESSAGE 'OLD';
  MESSAGE OldConsolidated.PKey || ',' || OldConsolidated.TextCol;
  MESSAGE 'NEW';
  MESSAGE NewRemote.PKey || ',' || NewRemote.TextCol;
  MESSAGE 'REMOTE';
  MESSAGE OldRemote.PKey || ',' || OldRemote.TextCol;
END;
```

The RESOLVE UPDATE trigger fires because the current value of the TextCol column at the consolidated database (`'Consolidated Update'`) does not match the value in the VALUES clause for the associated column (`'Initial'`).

This trigger results in a failure because the PKey column was not modified in the UPDATE statement executed on the remote, so there is no OldRemote.PKey value accessible from this trigger.

The CONFLICT function helps to avoid this error by returning the following values:

- If there is no OldRemote.PKey value, return FALSE.
- If there is an OldRemote.PKey value, but it matches OldConsolidated.PKey, return FALSE.
- If there is an OldRemote.PKey value, and it is different than OldConsolidated.PKey, return TRUE.

You can use the CONFLICT function to rewrite the trigger as follows and avoid the error:

```
CREATE TRIGGER ResolveUpdateAdmin
RESOLVE UPDATE ON Admin
```

```
REFERENCING OLD AS OldConsolidated
    NEW AS NewRemote
    REMOTE as OldRemote
FOR EACH ROW BEGIN
  message 'OLD';
  message OldConsolidated.PKey || ',' || OldConsolidated.TextCol;
  message 'NEW';
  message NewRemote.PKey || ',' || NewRemote.TextCol;
  message 'REMOTE';
  if CONFLICT( PKey ) then
    message OldRemote.PKey;
  end if;
  if CONFLICT( TextCol ) then
    message OldRemote.TextCol;
  end if;
END;
```

Related Information

CREATE TRIGGER Statement [page 1036]
Default Resolution for Update Conflicts

1.3.2.36 CONNECTION_EXTENDED_PROPERTY Function [String]

Returns the value of the given property. Allows an optional property-specific string parameter to be specified.

Syntax

```
CONNECTION_EXTENDED_PROPERTY( { property-id | property-name } [ , property-specific-argument [ , connection-id ] ] )
```

Parameters

property-id

The connection property ID.

property-name

The connection property name. The supported property names are:

CharSet

Returns the CHAR character set label for the connection as it is known by the specified standard. The possible values include: ASE, IANA, MIME, JAVA, WINDOWS, UTR22, IBM, and ICU. The default is IANA unless the database connection was made through TDS in which case ASE is the default.

NcharCharSet

Returns the NCHAR character set label for the connection as it is known by the specified standard. The possible values are the same as listed above for CharSet.

HasSecuredFeature Returns Yes if at least one of the features in the `property-specific-argument` argument is secured for the connection, otherwise returns No. Returns NULL if `property-specific-argument` is NULL.

Progress

Returns information about how long a statement has been executing. Specify a `property-specific-argument`, followed by `connection-id`, to return information specific to the statement's progress.

property-specific-argument

An optional property-specific string parameter associated with the following connection property:

HasSecuredFeature `feature-list`

Specify a list of features to determine whether at least one of these features is secured.

Progress

PercentComplete

Specify PercentComplete to obtain the percentage of the statement that has been processed.

Completed

Specify Completed to obtain the completed number of units.

Total

Specify Total to obtain the total number of units left to be processed.

Units

Specify Units to obtain the type of units left to be processed (pages, rows, or bytes).

Elapsed

Specify Elapsed to obtain the current elapsed time in milliseconds.

Remaining

Specify Remaining to obtain the estimated remaining time in milliseconds.

Raw

Specify Raw to obtain a string combining all of the above values in the order listed, separated by semicolons. For example, `43;9728;22230;pages;5025;6138`.

Formatted

Specify Formatted to obtain the human readable format. For example:

```
43% ( 9728 of 22230 pages ) complete after 00:00:05; estimated 00:00:06
remaining
```

The Remaining value may be empty if the remaining time has not yet been estimated, or if the number of units completed is greater than the original estimate.

For all property-specific arguments except Formatted, large byte values are never converted to kilobytes or megabytes.

connection-id

The connection ID number of a database connection. The ID number for the current connection is used if a value is not specified.

Returns

Returns extended connection properties. The returned value is a VARCHAR.

Remarks

Either the property ID or the property name must be specified.

The CONNECTION_EXTENDED_PROPERTY function is similar to the CONNECTION_PROPERTY function except that it allows an optional property-specific string parameter to be specified. The interpretation of the property-specific argument depends on the property ID or name specified in the first argument.

You can use the CONNECTION_EXTENDED_PROPERTY function to return the value for any connection property. However, extended information is only available for the extended properties.

Privileges

No privileges are required to execute this function for the current connection ID. To execute this function for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the CHAR character set of the current connection as it is known by the Java standard:

```
SELECT CONNECTION_EXTENDED_PROPERTY( 'charset', 'Java' );
```

Related Information

List of Connection Properties
progress_messages Option
CONNECTION_PROPERTY Function [System] [page 293]
DB_EXTENDED_PROPERTY Function [System] [page 329]
DB_PROPERTY Function [System] [page 337]

1.3.2.37 CONNECTION_PROPERTY Function [System]

Returns the value of a given connection property as a string.

Syntax

```
CONNECTION_PROPERTY( { property-id | property-name } [ , connection-id ] )
```

Parameters

property-id

The connection property ID.

property-name

The connection property name.

connection-id

The connection ID number of a database connection. The ID number for the current connection is used if a value is not specified.

Returns

VARCHAR, LONG VARCHAR

Remarks

Either the property ID or the property name must be specified.

Privileges

No privileges are required to execute this function for the current connection ID. To execute this function for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the number of prepared statements being maintained:

```
SELECT CONNECTION_PROPERTY( 'PrepStmt' );
```

Related Information

List of Connection Properties
PROPERTY_NUMBER Function [System] [page 496]

1.3.2.38 CONVERT Function [Data Type Conversion]

Returns an expression converted to a supplied data type.

Syntax

```
CONVERT( datatype , expression [ , format-style ] )
```

Parameters

datatype

The data type to convert the expression into. Set the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in a table or view, or to the data type of a variable.

expression

The expression to be converted.

format-style

The style code to apply to the output value. Use this parameter when converting strings to date or time data types, and vice versa. The table below shows the supported style codes, followed by a representation of the output format produced by that style code. The style codes are separated into two columns, depending on whether the century is included in the output format (for example, 06 versus 2006).

Style code 0 is used if an argument is not provided.

Without century (yy) style codes	With century (yyyy) style codes	Output format
-	0 or 100	Mmm dd yyyy hh:nnAA
1	101	mm/dd/yy[yy]
2	102	[yy]yy.mm.dd
3	103	dd/mm/yy[yy]
4	104	dd.mm.yy[yy]
5	105	dd-mm-yy[yy]
6	106	dd Mmm yy[yy]
7	107	Mmm dd, yy[yy]
8	108	hh:nn:ss
-	9 or 109	Mmm dd yyyy hh:nn:ss:sssAA
10	110	mm-dd-yy[yy]
11	111	[yy]yy/mm/dd
12	112	[yy]yymmdd
-	13 or 113	dd Mmm yyyy hh:nn:ss:sss (24 hour clock, Europe default + milliseconds, 4-digit year)
-	14 or 114	hh:nn:ss:sss (24 hour clock)
-	20 or 120	yyyy-mm-dd hh:nn:ss (24-hour clock, ODBC canonical, 4-digit year)
-	21 or 121	yyyy-mm-dd hh:nn:ss.sss (24 hour clock, ODBC canonical with milliseconds, 4-digit year)

Returns

Depends on the data type specified.

Remarks

The CONVERT function can be used to convert a string to a DATE, TIME, or TIMESTAMP data type, provided that there is no ambiguity when parsing the string. If `format-style` is specified, then the database server may use it as a hint on how to parse the string. The database server returns an error if it cannot parse the string unambiguously.

UltraLite: This function is similar to the CAST function but allows you to specify a format style to assist with date and time data type conversions.

Standards

ANSI/ISO SQL Standard

The CONVERT function is defined in the ANSI/ISO SQL Standard. However, in the Standard the purpose of CONVERT is to perform a transcoding of the input string expression to a different character set, which is implemented in the software as the CSCONVERT function.

Example

The following statements illustrate the use of format style:

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 104 ) FROM GROUPO.SalesOrders;
```

OrderDate
16.03.2000
20.03.2000
23.03.2000
25.03.2000
...

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 7 ) FROM GROUPO.SalesOrders;
```

OrderDate
Mar 16, 00
Mar 20, 00
Mar 23, 00
Mar 25, 00
...

The following statement illustrates conversion to an integer and returns the value 5:

```
SELECT CONVERT( integer, 5.2 );
```

The following statement converts a value to the data type defined for the BirthDate column (DATE data type) of the Employees table:

```
SELECT CONVERT ( Employees.BirthDate%TYPE, '1966-10-30' );
```

Related Information

date_format Option
CAST Function [Data Type Conversion] [page 275]
CSCONVERT Function [String] [page 309]

1.3.2.39 CORR Function [Aggregate]

Returns the correlation coefficient of a set of number pairs.

Syntax

```
CORR( dependent-expression , independent-expression )
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

Both `dependent-expression` and `independent-expression` are numeric. The function is applied to the set of (`dependent-expression`, `independent-expression`) after eliminating the pairs for which either `dependent-expression` or `independent-expression` is NULL. The following computation is made:

COVAR_POP (`y`, `x`) / STDDEV_POP (`y`) * STDDEV_POP (`x`)

where `y` represents the `dependent-expression` and `x` represents the `independent-expression`.

Standards

ANSI/ISO SQL Standard

The CORR function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following example performs a correlation to discover whether age is associated with income level and returns the value 0.44022675645996:

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM
GROUPO.Employees;
```

Related Information

Aggregate Functions [page 208]
COVAR_POP Function [Aggregate] [page 305]
STDDEV_POP Function [Aggregate] [page 568]

1.3.2.40 COS Function [Numeric]

Returns the cosine of the angle in radians given by its argument.

Syntax

```
COS( numeric-expression )
```

Parameters

numeric-expression

The angle, in radians.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value of the cosine of an angle 0.52 radians:

```
SELECT COS( 0.52 );
```

Related Information

ACOS Function [Numeric] [page 241]
COT Function [Numeric] [page 299]
SIN Function [Numeric] [page 553]
TAN Function [Numeric] [page 587]

1.3.2.41 COT Function [Numeric]

Returns the cotangent of the angle in radians given by its argument.

Syntax

```
COT( numeric-expression )
```

Parameters

numeric-expression

The angle, in radians.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the cotangent value of 0.52:

```
SELECT COT( 0.52 );
```

Related Information

COS Function [Numeric] [page 298]
SIN Function [Numeric] [page 553]
TAN Function [Numeric] [page 587]

1.3.2.42 COUNT Function [Aggregate]

Counts the number of rows in a group depending on the specified parameters.

Syntax

Expressions

```
COUNT( [ * | [ ALL | DISTINCT ] expression ] )
```

Window function

```
COUNT( [ * | [ ALL ] expression ] ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

UltraLite expressions

```
COUNT( [ * | [ DISTINCT ] expression ] )
```

Parameters

*

Return the number of rows in each group. COUNT(*) and COUNT() are semantically equivalent.

[ALL] expression

Return the number of rows in each group where the value of `expression` is not NULL.

DISTINCT expression

Return the number of distinct values of `expression` for all of the rows in each group where `expression` is not NULL.

UltraLite expression

Return the number of rows in each group where the value of `expression` is not null.

p

Returns

The COUNT function returns a value of type INT.

COUNT never returns the value NULL. If a group contains no rows, or if there are no non-NULL values of `expression` in a group, then COUNT returns 0.

Remarks

In SQL Anywhere, the COUNT function returns a maximum value of 2147483647. Use the COUNT_BIG function when counting large result sets, the result might have more rows, or there is a possibility of overflow.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

Core Feature. When used as a window function, COUNT comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the COUNT function, combined with an outer reference.

Example

The following statement returns each unique city, and the number of employees working in that city:

```
SELECT City, COUNT( * ) FROM GROUPO.Employees GROUP BY City;
```

The following statement returns each unique city, and the number of managers working in that city:

```
SELECT City, COUNT( DISTINCT ManagerID ) FROM GROUPO.Employees GROUP BY City;
```

Related Information

Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
COUNT(*)
WINDOW Clause [page 1489]
AVG Function [Aggregate] [page 254]
SUM Function [Aggregate] [page 581]
COUNT_BIG Function [Aggregate] [page 303]
Troubleshooting Database Upgrades: Aggregate Functions and Outer References

1.3.2.43 COUNT_BIG Function [Aggregate]

Counts the number of rows in a group depending on the specified parameters.

Syntax

Expressions

```
COUNT_BIG( [ * | [ ALL | DISTINCT ] expression ] )
```

Window function

```
COUNT_BIG( [ * | [ ALL ] expression ] ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

*****

Return the number of rows in each group. COUNT_BIG(*) and COUNT_BIG() are semantically equivalent.

[ALL] expression

Return the number of rows in each group where the value of `expression` is not NULL.

DISTINCT expression

Return the number of distinct values of `expression` for all of the rows in each group where `expression` is not NULL.

Returns

COUNT_BIG returns a value of type BIGINT.

COUNT_BIG never returns the value NULL. If a group contains no rows, or if there are no non-NULL values of `expression` in a group, then COUNT_BIG returns 0.

Remarks

It is recommended that you use the COUNT_BIG function when counting large result sets, the result might have more rows, or there is a possibility of overflow. Otherwise, use the COUNT function, which has a maximum value of 2147483647.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

Not in the standard.

The software does not support optional Feature F442, "Mixed column references in set functions". The software also does not permit the arguments of an aggregate function to include both a column reference from the query block containing the COUNT_BIG function, combined with an outer reference.

Example

The following statement returns each unique city, and the number of employees working in that city:

```
SELECT City, COUNT_BIG( * ) FROM GROUPO.Employees GROUP BY City;
```

The following statement returns each unique city, and the number of managers working in that city:

```
SELECT City, COUNT_BIG( DISTINCT ManagerID ) FROM GROUPO.Employees GROUP BY City;
```

Related Information

Window Definition: Inlining Using the OVER Clause and WINDOW Clause
Window Functions
AVG Function [Aggregate] [page 254]
SUM Function [Aggregate] [page 581]
COUNT Function [Aggregate] [page 300]
Troubleshooting Database Upgrades: Aggregate Functions and Outer References

1.3.2.44 COUNT_SET_BITS Function [Bit Array]

Returns a count of the number of bits set to 1 (TRUE) in the array.

Syntax

```
COUNT_SET_BITS( bit-expression )
```

Parameters

bit-expression

The bit array for which to determine the set bits.

Returns

UNSIGNED INT

Remarks

Returns NULL if `bit-expression` is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 4:

```
SELECT COUNT_SET_BITS( '00110011' );
```

The following statement returns the value 12:

```
SELECT COUNT_SET_BITS( '0011001111111111' );
```

1.3.2.45 COVAR_POP Function [Aggregate]

Returns the population covariance of a set of number pairs.

Syntax

Expression

```
COVAR_POP( dependent-expression , independent-expression )
```

Window function

```
COVAR_POP( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

Both `dependent-expression` and `independent-expression` are numeric. The function is applied to the set of (`dependent-expression`, `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The following computation is then made:

```
( SUM( y * x ) - SUM( x ) * SUM( y ) / n ) / n
```

where `y` represents the `dependent-expression` and `x` represents the `independent-expression`.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

The COVAR_POP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following example measures the strength of association between employees' age and salary. This function returns the value 73785.84005866687.

```
SELECT COVAR_POP( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
COVAR_SAMP Function [Aggregate] [page 307]
SUM Function [Aggregate] [page 581]

1.3.2.46 COVAR_SAMP Function [Aggregate]

Returns the sample covariance of a set of number pairs.

Syntax

Expression

```
COVAR_SAMP( dependent-expression , independent-expression )
```

Window function

```
COVAR_SAMP( dependent-expression , independent-expression )
OVER ( window-spec )

window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

Both `dependent-expression` and `independent-expression` are numeric. The function is applied to the set of (`dependent-expression`, `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

The COVAR_SAMP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following example returns the value 74782.9460054052:

```
SELECT COVAR_SAMP( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
COVAR_POP Function [Aggregate] [page 305]
SUM Function [Aggregate] [page 581]

1.3.2.47 CSCONVERT Function [String]

Converts strings between character sets.

Syntax

```
CSCONVERT(
string-expression ,
target-charset-string
[ , source-charset-string [ , options ] ]
)
```

Parameters

string-expression

The string to be converted.

target-charset-string

The destination character set. `target-charset-string` can be any of the supported character set labels. It can also be:

db_charset Equivalent to char_charset. Specify this to use the CHAR character set used by the database.

os_charset

Specify this to use the character set used by the operating system that is hosting the database server.

char_charset

Equivalent to db_charset. Specify this to use the CHAR character set used by the database.

nchar_charset

Specify this to use the NCHAR character set used by the database.

source-charset-string

The character set used for `string-expression`. The default is db_charset (the database character set). `source-charset-string` can be any of the supported character set labels. It can also be:

db_charset Equivalent to char_charset. Specify this to use the CHAR character set used by the database.

os_charset

Specify this to use the character set used by the operating system that is hosting the database server.

char_charset

Equivalent to db_charset. Specify this to use the CHAR character set used by the database.

nchar_charset

Specify this to use the NCHAR character set used by the database.

options

You can specify either or both of the following options:

Read or write a byte order mark (BOM)

Specify **read_bom=on** or **read_bom=off** to turn on or off reading byte order marks. Specify **write_bom=on** or **write_bom=off** to turn on or off writing byte order marks. By default, the behavior is read_bom=on and write_bom=off.

Returns

LONG BINARY

Remarks

You can view the list of supported character sets by running the following command:

```
dbinit -le
```

Standards

ANSI/ISO SQL Standard

Not in the standard. In the ANSI/ISO SQL Standard, conversion of string data from one charset to another is accomplished with the CONVERT function (not to be confused with the CONVERT function provided in the software) which has different arguments than CSCONVERT.

Example

This fragment converts the mytext column from the Traditional Chinese character set to the Simplified Chinese character set:

```
SELECT CSCONVERT( mytext, 'cp936', 'cp950' )
FROM mytable;
```

This fragment treats the mytext column as if it were encoded in 'cp950' and converts it to 'cp936'. Note that mytext may or may not actually be encoded in cp950 (that may depend on the CHAR collation and/or the datatype of the mytext column):

```
SELECT CSCONVERT( mytext, 'cp936' )
FROM mytable;
```

If a file name is stored in the database, it is stored in the database character set. If the server will read from or write to a file whose name is stored in a database (for example, in an external stored procedure), the file name

must be explicitly converted to the operating system character set before the file can be accessed. File names stored in the database and retrieved by the client are converted automatically to the client character set, so explicit conversion is not necessary.

This fragment converts the value in the filename column from the database character set to the operating system character set:

```
SELECT CSCONVERT( filename, 'os_charset' )
FROM mytable;
```

A table contains a list of file names. An external stored procedure takes a file name from this table as a parameter and reads information directly out of that file. The following statement works when character set conversion is not required:

```
SELECT MYFUNC( filename )
FROM mytable;
```

The `mytable` clause indicates a table with a filename column. However, if you need to convert the file name to the character set of the operating system, you would use the following statement:

```
SELECT MYFUNC( csconvert( filename, 'os_charset' ) )
FROM mytable;
```

Related Information

Supported Character Sets
String Functions [page 222]

1.3.2.48 CUME_DIST Function [Ranking]

Computes the relative position of one value among a group of rows.

Syntax

```
CUME_DIST( ) OVER ( window-spec )

window-spec : see the Remarks section below
```

Returns

A DOUBLE value between 0 and 1

Remarks

Composite sort keys are not currently allowed in the CUME_DIST function. You can use composite sort keys with any of the other rank functions.

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause.

Standards

ANSI/ISO SQL Standard

The CUME_DIST function comprises part of optional SQL Language Feature T612, "Advanced OLAP operations".

Example

The following example returns a result set that provides a cumulative distribution of the salaries of employees who live in California:

```
SELECT DepartmentID, Surname, Salary,
CUME_DIST() OVER (PARTITION BY DepartmentID
ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees
WHERE State IN ('CA');
```

Here is the result set:

DepartmentID	Surname	Salary	Rank
200	Savarino	72300.000	0.333333333333333
200	Clark	45000.000	0.666666666666667
200	Overbey	39300.000	1

Related Information

Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
DENSE_RANK Function [Ranking] [page 346]
PERCENT_RANK Function [Ranking] [page 485]
RANK Function [Ranking] [page 501]

1.3.2.49 DATALENGTH Function [System]

Returns the length, in bytes, of the underlying storage for the result of an expression.

Syntax

```
DATALENGTH( expression )
```

Parameters

expression

Usually a column name. If `expression` is a string constant, you must enclose it in quotes.

Returns

UNSIGNED INT

Remarks

The return values of the DATALENGTH function are as follows:

Data type	DATALENGTH
BIT	1
TINYINT	1
SMALLINT	2
INTEGER	4
BIGINT	8
REAL	4
DOUBLE	8
TIME	8
DATE	4
TIMESTAMP	8
DATETIME	8
TIMESTAMP WITH TIME ZONE	29
UNIQUEIDENTIFIER	16

Data type	DATALENGTH
CHAR	Length of the data
VARCHAR	Length of the data
BINARY	Length of the data
VARBINARY	Length of the data
NCHAR	Length of the data
NVARCHAR	Length of the data
TEXT	Length of the data
NTEXT	Length of the data
IMAGE	Length of the data
XML	Length of the data

In SQL Anywhere, this function supports NCHAR inputs and outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the length of the longest string in the CompanyName column:

```
SELECT MAX( DATALENGTH( CompanyName ) )
FROM GROUPO.Customers;
```

The following statement returns the length of the string '8sdofinsv8s7a7s7gehe4h':

```
SELECT DATALENGTH( '8sdofinsv8s7a7s7gehe4h' );
```

Related Information

SQL Data Types [page 129]

1.3.2.50 DATE Function [Date and Time]

Converts the expression into a date, and removes any hours, minutes, or seconds.

Syntax

```
DATE( expression )
```

Parameters

expression

The value to be converted to date format, typically a string.

Returns

DATE

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 1999-01-02 as a date:

```
SELECT DATE( '1999-01-02 21:20:53' );
```

The following statement returns the create dates of all the objects listed in the SYSOBJECT system view:

```
SELECT DATE( creation_time ) FROM SYS.SYSOBJECT;
```

Related Information

nearest_century Option
date_order Option

Expressions in SQL Statements [page 34]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.3.2.51 DATEADD Function [Date and Time]

Returns a TIMESTAMP or TIMESTAMP WITH TIME ZONE value produced by adding a date part to its argument.

Syntax

```
DATEADD( date-part , integer-expression , timestamp-expression )
```

```
date-part :
year | quarter | month | week | day | dayofyear | hour | minute | second | millisecond
| microsecond
```

Parameters

date-part

The date part that `integer-expression` represents.

integer-expression

The number of `date-part` values to be added to `timestamp-expression`. `integer-expression` can be any numeric type, but its value is truncated to an INTEGER. This value can be positive, zero, or negative.

timestamp-expression

The TIMESTAMP or TIMESTAMP WITH TIME ZONE value to be modified.

Returns

TIMESTAMP WITH TIME ZONE if `timestamp-expression` is a TIMESTAMP WITH TIME ZONE; otherwise TIMESTAMP.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the TIMESTAMP value 2016-05-02 00:00:00.000:

```
SELECT DATEADD( month, 12, '2015/05/02' );
```

The following statement returns the TIMESTAMP value 2015-05-02 04:00:00.000:

```
SELECT DATEADD( hour, 4, '2015/05/02' );
```

You can specify a minus sign to subtract from a date or time. For example, to get a timestamp from 31 days ago, you can execute the following:

```
SELECT DATEADD( day, -31, NOW() );
```

The following statement returns the TIMESTAMP WITH TIME ZONE value 2015-05-06 11:33:00.000+04:00:

```
SELECT DATEADD( day, 4, CAST( '2015/05/02 11:33:00.000000+04:00' as TIMESTAMP
WITH TIME ZONE ) );
```

Related Information

Specifying Date Parts [page 214]

1.3.2.52 DATEDIFF Function [Date and Time]

Returns the interval between two dates.

Syntax

```
DATEDIFF( date-part , date-expression-1 , date-expression-2 )
```

```
date-part :
year | quarter | month | week | day | dayofyear | hour | minute | second | millisecond
| microsecond
```

Parameters

date-part

Specifies the date part in which the interval is to be measured.

date-expression-1

The starting date for the interval. This value is subtracted from `date-expression-2` to return the number of `date-parts` between the two arguments.

date-expression-2

The ending date for the interval. `Date-expression-1` is subtracted from this value to return the number of `date-parts` between the two arguments.

Returns

INT with year, quarter, month, week, day, and dayofyear. BIGINT with hour, minute, second, millisecond, and microsecond.

Remarks

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to (date-expression-2 - date-expression-1), in date parts.

The DATEDIFF function results are truncated, not rounded, when the result is not an even multiple of the date part.

When you use **`day`** as the date part, the DATEDIFF function returns the number of midnights between the two times specified, including the second date but not the first.

When you use **`month`** as the date part, the DATEDIFF function returns the number of first-of-the-months between two dates, including the second date but not the first.

When you use **`week`** as the date part, the DATEDIFF function returns the number of Sundays between the two dates, including the second date but not the first.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns 1:

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' );
```

The following statement returns 102:

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' );
```

The following statement returns 0:

```
SELECT DATEDIFF( day, '00:00', '23:59' );
```

The following statement returns 4:

```
SELECT DATEDIFF( day,
   '1999/07/19 00:00',
   '1999/07/23 23:59' );
```

The following statement returns 0:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' );
```

The following statement returns 1:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' );
```

The following example shows how to use the DATEDIFF function to return the number of milliseconds to do a 3-way join with the GROUPO.Customers table. The output is sent to the database server window:

```
BEGIN
    DECLARE startTime, endTime TIMESTAMP;
    DECLARE rowCount INT;

    SET startTime = CURRENT TIMESTAMP;
    SELECT count(*) INTO rowCount FROM GROUPO.Customers AS T1, GROUPO.Customers
AS T2, GROUPO.Customers AS T3;
    SET endTime = CURRENT TIMESTAMP;

    MESSAGE 'Time to count rows: ' || DATEDIFF( MILLISECOND, startTime,
endTime ) || ' ms';
END
```

Related Information

Specifying Date Parts [page 214]
DATEADD Function [Date and Time] [page 316]

1.3.2.53 DATEFORMAT Function [Date and Time]

Returns a string representing a date expression in the specified format.

Syntax

```
DATEFORMAT( datetime-expression , string-expression )
```

Parameters

datetime-expression

The datetime to be converted.

string-expression

The format of the converted date.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Returns

VARCHAR

Remarks

Any allowable date format can be used for the string-expression.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value Jan 01, 1989:

```
SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );
```

Related Information

timestamp_format Option

1.3.2.54 DATENAME Function [Date and Time]

Returns the name of the specified part (such as the month June) of a TIMESTAMP or TIMESTAMP WITH TIME ZONE value, as a character string.

Syntax

```
DATENAME( date-part , timestamp-expression )
```

Parameters

date-part

The date part to be named.

timestamp-expression

The TIMESTAMP or TIMESTAMP WITH TIME ZONE value for which the date part name is to be returned. For meaningful results, `timestamp-expression` should contain the requested `date-part`.

Returns

VARCHAR

Remarks

The DATENAME function returns a string, even if the result is numeric, such as 23 for the day.

In SQL Anywhere, English names are returned for an English locale, other names are returned when the locale is not English. For example, use the Language (LANG) connection parameter to specify a different language.

When the date part TZOffset (TZ) is specified, DATENAME returns the offset as a string of the form: { + | - }hh:nn.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In an English locale, the following statement returns the value `May`:

```
SELECT DATENAME( month, '1987/05/02' );
```

On SQL Anywhere in a German locale, the value returned is `Mai`. In a Spanish locale, the value returned is `Mayo`. Several locales are supported.

The following statement returns the value `-05:00`:

```
SELECT DATENAME( TZ, CAST('2016/02/03 12:02:00-5:00' AS TIMESTAMP WITH TIME
ZONE) ) AS TZOffset;
```

Related Information

Specifying Date Parts [page 214]
DATEPART Function [Date and Time] [page 322]
MONTHNAME Function [Date and Time] [page 463]
Language (LANG) Connection Parameter

1.3.2.55 DATEPART Function [Date and Time]

Returns a portion of a TIMESTAMP or TIMESTAMP WITH TIME ZONE value.

Syntax

```
DATEPART( date-part , timestamp-expression )
```

Parameters

date-part

The date part to be returned.

timestamp-expression

The TIMESTAMP or TIMESTAMP WITH TIME ZONE value for which the part is to be returned.

Returns

INT

Remarks

For meaningful results `timestamp-expression` should contain the required `date-part` portion.

The numbers that correspond to week days depend on the setting of the first_day_of_week database option. By default, first_day_of_week is 7 which means Sunday.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 5:

```
SELECT DATEPART( month , '1987/05/02' );
```

For TIMESTAMP WITH TIME ZONE strings, the string value must be cast as follows to ensure conversion to the correct type.

```
SELECT DATEPART( TZOffset, CAST('2015-07-01 12:34:56.789000 +05:30' AS TIMESTAMP
WITH TIME ZONE) );
```

The following example creates a table, TableStatistics, and inserts into it the total number of sales orders per year as stored in the SalesOrders table:

```
CREATE TABLE TableStatistics (
   ID INTEGER NOT NULL DEFAULT AUTOINCREMENT,
   Year INT,
   NumberOrders INT );
INSERT INTO TableStatistics ( Year, NumberOrders )
   SELECT DATEPART( Year, OrderDate ), COUNT(*)
   FROM GROUPO.SalesOrders
   GROUP BY DATEPART( Year, OrderDate );
```

Related Information

Specifying Date Parts [page 214]
first_day_of_week Option
SET Statement [T-SQL] [page 1375]
EXTRACT Function [Date and Time] [page 383]
MICROSECOND Function [Date and Time] [page 453]
MILLISECOND Function [Date and Time] [page 454]

1.3.2.56 DATETIME Function [Date and Time]

Converts an expression into a TIMESTAMP value.

Syntax

```
DATETIME( expression )
```

Parameters

expression

The expression to be converted. It is generally a string.

Returns

TIMESTAMP

Remarks

Attempts to convert numerical values return an error.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns a timestamp with value 1998-09-09 12:12:12.000:

```
SELECT DATETIME( '1998-09-09 12:12:12.000' );
```

Related Information

Expressions in SQL Statements [page 34]
CAST Function [Data Type Conversion] [page 275]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIMEOFFSET Data Type [page 171]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.3.2.57 DAY Function [Date and Time]

Returns the day of the month of its argument as an integer between 1 and 31.

Syntax

```
DAY( date-expression )
```

Parameters

date-expression

The date as a DATE data type.

Returns

SMALLINT

Remarks

The DAY function returns an integer between 1 and 31, corresponding to the day of the month in the argument.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 12:

```
SELECT DAY( '2001-09-12' );
```

1.3.2.58 DAYNAME Function [Date and Time]

Returns the name of the day of the week from a date.

Syntax

```
DAYNAME( date-expression )
```

Parameters

date-expression

The date.

Returns

VARCHAR

Remarks

The names are returned as: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.

SQL Anywhere returns English names for an English locale, and returns other names when the locale is not English. For example, the Language (LANG) connection parameter can be used to specify a different language.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In an English locale, the following statement returns the value `Saturday`:

```
SELECT DAYNAME( '1987/05/02' );
```

In a German locale, the value returned is `Samstag`. In an Italian locale, the value returned is `sabato`. Several locales are supported.

Related Information

Language (LANG) Connection Parameter

1.3.2.59 DAYS Function [Date and Time]

Manipulates a TIMESTAMP or returns the number of days between two TIMESTAMP values.

Syntax

Return number of days between 0000-02-29 and a TIMESTAMP value

```
DAYS( timestamp-expression )
```

Return number of days between two TIMESTAMP values

```
DAYS( timestamp-expression , timestamp-expression )
```

Add time to a TIMESTAMP

```
DAYS( timestamp-expression , integer-expression )
```

Parameters

timestamp-expression

A TIMESTAMP value.

integer-expression

The number of days to be added to the `timestamp-expression`. If the `integer-expression` is negative, the appropriate number of days is subtracted from `timestamp-expression`. If you supply an integer expression, the `timestamp-expression` must be explicitly cast as a TIME, DATE or TIMESTAMP. If `timestamp-expression` is a TIME value, the current date is assumed.

Returns

TIMESTAMP when adding time to a timestamp; otherwise, INTEGER.

Remarks

The result of the DAYS function depends on its arguments. The DAYS function ignores hours, minutes, and seconds in its arguments.

Return number of days since 0000-02-29

If you pass a single `timestamp-expression` to the DAYS function, it will return the number of days between 0000-02-29 and `timestamp-expression` as an INTEGER.

> **i** Note
>
> 0000-02-29 is not meant to imply an actual date; it is the default date used by the DAYS function.

Return number of days between two TIMESTAMP values

If you pass two TIMESTAMP values to the DAYS function, the function returns the integer number of days between them.

You can also use the DATEDIFF function to get the interval between two dates.

Add time to a TIMESTAMP

If you pass a TIMESTAMP value and an integer to the DAYS function, the function returns the TIMESTAMP result of adding the integer number of days to the `timestamp-expression` argument.

You can also use the DATEADD function to add a date part to a TIMESTAMP.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the integer 729889:

```
SELECT DAYS( '1998-07-13 06:07:12' );
```

The following statements return the integer value -366, indicating that the second DATE value is 366 days before the first. It is recommended that you use the second example (DATEDIFF):

```
SELECT DAYS( '1998-07-13 06:07:12',
             '1997-07-12 10:07:12' );
```

```
SELECT DATEDIFF( day,
   '1998-07-13 06:07:12',
   '1997-07-12 10:07:12' );
```

The following statements return the TIMESTAMP value 1999-07-14 00:00:00.000. It is recommended that you use the second example (DATEADD):

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 );
```

```
SELECT DATEADD( day, 366, '1998-07-13' );
```

Related Information

DATEDIFF Function [Date and Time] [page 317]
DATEADD Function [Date and Time] [page 316]
CAST Function [Data Type Conversion] [page 275]

1.3.2.60 DB_EXTENDED_PROPERTY Function [System]

Returns the value of the given property. Allows an optional property-specific string parameter to be specified.

Syntax

```
DB_EXTENDED_PROPERTY( { property-id | property-name } [ , property-specific-
argument [ , database-id | database-name ] ] )
```

Parameters

property-id

The database property ID to query.

property-name

The database property name to query.

property-specific-argument

The following database properties allow you to specify additional arguments, as noted below, to return specific information about the property.

CatalogCollation, Collation, NcharCollation

When querying these properties, the following values can be specified as a `property-specific-argument` to return information specific to the collation:

AccentSensitivity

Returns the accent sensitivity setting for the collation. For example, the following statement returns the accent sensitivity setting for the NCHAR collation:

```
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'AccentSensitivity');
```

Possible return values are: NULL, Ignore, Respect, and French.

CaseSensitivity

Returns the case sensitivity setting for the collation. Possible return values are: NULL, Ignore, Respect, UpperFirst, and LowerFirst.

PunctuationSensitivity

Returns the punctuation sensitivity setting for the collation. Possible return values are: NULL, Ignore, Primary, and Quaternary.

Properties

Returns a string containing all the tailoring options specified for the collation.

Specification

Returns a string containing the full collation specification used for the collation.

CharSet

Specify the name of a standard to obtain the default CHAR character set label for the standard. Possible values you can specify are: ASE, IANA, MIME, JAVA, WINDOWS, UTR22, IBM, and ICU. If no standard is specified, IANA is used as the default, unless the database connection was made through TDS, in which case ASE is the default.

DBFileFragments

Specify the name of a dbspace, or the file ID for the dbspace, to obtain the number of file fragments. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

DriveBus

(Microsoft Windows only) Specify the name of a dbspace, or the file ID for the dbspace, to obtain the configuration of the drive on which it resides. DriveBus returns BusType from an

IOCTL_STORAGE_QUERY_PROPERTY call. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

DriveModel

(Microsoft Windows only) Specify the name of a dbspace, or the file ID for the dbspace, to obtain the model of the drive on which it resides. DriveModel returns the concatenation of the VendorId, ProductId, and ProductRevision strings from an IOCTL_STORAGE_QUERY_PROPERTY call. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

DriveType

Specify the name of a dbspace, or the file ID for the dbspace, to obtain its drive type. The value returned is one of the following: CD, FIXED, RAMDISK, REMOTE, REMOVABLE, or UNKNOWN. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

File

Specify the name of a dbspace, or the file ID for the dbspace, to obtain the file name of the database root file, including the path. Specify *'translog'* to obtain the path and file name of the transaction log file, and *'translogmirror'* to obtain the path and file name of the transaction log mirror file. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

FileSize

Specify the name of a dbspace, or the file ID for the dbspace, to obtain the size of the specified file in pages. You can also specify *'temporary'* to return the size of the temporary dbspace, *'translog'* to return the size of the transaction log file, and *'translogmirror'* to return the size of the transaction log file mirror. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

FreePages

Specify the name of a dbspace, or the file ID for the dbspace, to obtain the number of free pages. You can also specify temporary to return the number of free pages in the temporary dbspace, or translog to return the number of free pages in the transaction log file. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

IOParallelism

Specify the name of a dbspace, or the file ID for the dbspace, to obtain the estimated number of simultaneous I/O operations supported by the dbspace. If you do not specify the dbspace name or file ID, then the system dbspace is used. If the specified dbspace name or ID does not exist for the database to which you are connected, then the function returns NULL.

MirrorServerState

Specify a server name to determine the connection status of the mirror server. The value returned is *NULL* when the database is not mirrored; *connected* when there is a connection from this server to a specified server and a connection from the specified server to this server; *disconnected* when there are no connections between this server and the specified server; *incoming only* when there is a connection

from the specified server to this server; and *outgoing only* when there is a connection from this server to the specified server.

MirrorState

Specify a server name to determine the synchronization status of the mirror server. The value returned is *synchronizing* when the mirror server is not connected, or has not yet read all the primary server's log pages, or if the synchronization mode is asynchronous. The value returned is *synchronized* when the mirror server is connected and has all of the changes that have been committed on the primary server. If the database is not being mirrored the value returned is *NULL*.

NcharCharSet

Specify the name of a standard to obtain the default NCHAR character set encoding label for that standard. Possible values you can specify are: ASE, IANA, MIME, JAVA, WINDOWS, UTR22, IBM, and ICU. If no standard is specified, IANA is used as the default, unless the database connection was made through TDS, in which case ASE is the default.

NextScheduleTime

Specify an event name to obtain its next scheduled execution time.

database-id

The database ID number, as returned by the DB_ID function. Typically, the database name is used.

database-name

The name of the database, as returned by the DB_NAME function.

Returns

VARCHAR

Remarks

The DB_EXTENDED_PROPERTY function is similar to the DB_PROPERTY function except that it allows an optional `property-specific-argument` string parameter to be specified. The interpretation of `property-specific-argument` depends on the property ID or name specified in the first argument.

The current database is used if the third argument is omitted.

When comparing catalog strings such as table names and procedure names, the database server uses the CHAR collation. For the UCA collation, the catalog collation is the same as the CHAR collation but with the tailoring changed to be case-insensitive, accent-insensitive and with punctuation sorted in the primary level. For legacy collations, the catalog collation is the same as the CHAR collation but with the tailoring changed to be case-insensitive. While you cannot explicitly specify the tailoring used for the catalog collation, you can query the Specification property to obtain the full collation specification used by the database server for comparing catalog strings. Querying the Specification property can be useful if you need to exploit the difference between the CHAR and catalog collations. For example, suppose you have a punctuation-insensitive CHAR collation and you want to execute an upgrade script that defines a procedure called my_procedure, and

that also attempts to delete an old version named myprocedure. The following statements cannot achieve the desired results because my_procedure is equivalent to myprocedure, using the CHAR collation:

```
CREATE PROCEDURE my_procedure( ) ...;
IF EXISTS ( SELECT * FROM SYS.SYSPROCEDURE WHERE proc_name = 'myprocedure' )
THEN DROP PROCEDURE myprocedure
END IF;
```

Instead, you could execute the following statements to achieve the desired results:

```
CREATE PROCEDURE my_procedure( ) ...;
IF EXISTS ( SELECT * FROM SYS.SYSPROCEDURE
   WHERE COMPARE( proc_name, 'myprocedure',
DB_EXTENDED_PROPERTY( 'CatalogCollation', 'Specification' ) ) = 0 )
THEN DROP PROCEDURE myprocedure
END IF;
```

Privileges

No privileges are required to execute this function for the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the location of the current database:

```
SELECT DB_EXTENDED_PROPERTY( 'File' );
```

The following statement returns the file size of the system dbspace, in pages:

```
SELECT DB_EXTENDED_PROPERTY( 'FileSize' );
```

The following statement returns the file size of the transaction log, in pages:

```
SELECT DB_EXTENDED_PROPERTY( 'FileSize', 'translog' );
```

The following statement returns the character set labels for two different databases started by the database server:

```
SELECT DB_EXTENDED_PROPERTY( 'CharSet', 'IANA', DB_NAME() ),
```

```
        DB_EXTENDED_PROPERTY( 'CharSet', 'IANA', 'JapanDemo' );
```

The following statement returns the case sensitivity setting for the NCHAR collation:

```
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation',' CaseSensitivity' );
```

The following statement returns the tailoring options specified for the database CHAR collation:

```
SELECT DB_EXTENDED_PROPERTY ( 'Collation', 'Properties' );
```

The following statement returns the full collation specification for the database NCHAR collation:

```
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'Specification' );
```

The following statement returns the connection status of the mirror server Test:

```
SELECT DB_EXTENDED_PROPERTY( 'MirrorServerState', 'Test' );
```

The following statement returns the synchronization status of the mirror server Test:

```
SELECT DB_EXTENDED_PROPERTY( 'MirrorState', 'Test' );
```

Related Information

Collation Tailoring Options
DB_ID Function [System] [page 334]
DB_NAME Function [System] [page 336]
List of Database Server Properties
CONNECTION_PROPERTY Function [System] [page 293]
CONNECTION_EXTENDED_PROPERTY Function [String] [page 290]

1.3.2.61 DB_ID Function [System]

Returns the database ID number.

Syntax

```
DB_ID( [ database-name ] )
```

Parameters

database-name

A string containing the database name. If no `database-name` is supplied, the ID number of the current database is returned.

Returns

INT

Remarks

None

Privileges

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 0, when executed against the sample database as the sole database on the server:

```
SELECT DB_ID( 'demo' );
```

The following statement returns the value 0 if executed against the only running database:

```
SELECT DB_ID( );
```

Related Information

global_database_id Option

1.3.2.62 DB_NAME Function [System]

Returns the name of a database with a given ID number.

Syntax

```
DB_NAME( [ database-id ] )
```

Parameters

database-id

The ID of the database. The `database-id` must be a numeric expression.

Returns

VARCHAR

Remarks

If no database ID is supplied, the name of the current database is returned.

Privileges

No privileges are required to execute this function for the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the database name demo when executed against the sample database as the sole database on the server:

```
SELECT DB_NAME( 0 );
```

Related Information

sa_db_list System Procedure [page 1565]
NEXT_DATABASE Function [System] [page 471]

1.3.2.63 DB_PROPERTY Function [System]

Returns the value of the specified database property.

Syntax

```
DB_PROPERTY( { property-id | property-name } [ , database-id | database-name ] )
```

UltraLite:

```
DB_PROPERTY( property-name )
```

Parameters

property-id

The database property ID.

property-name

The database property name.

database-id

The database ID number, as returned by the DB_ID function. Typically, the database name is used.

database-name

The name of the database, as returned by the DB_NAME function.

Returns

VARCHAR, LONG VARCHAR

Remarks

Returns a string.

The current database is used if the second argument is omitted.

UltraLite: To set an option in UltraLite, use the SET OPTION statement or your component's API-specific Set Database Option method.

Privileges

No privileges are required to execute this function for the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

UltraLite: These privileges do not apply to UltraLite.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the page size of the current database, in bytes:

```
SELECT DB_PROPERTY( 'PageSize' );
```

Related Information

DB_ID Function [System] [page 334]
DB_NAME Function [System] [page 336]
List of Database Properties

PROPERTY Function [System] [page 490]

1.3.2.64 DECOMPRESS Function [String]

Decompresses the string and returns a LONG BINARY value.

Syntax

```
DECOMPRESS( string-expression [ , compression-algorithm-alias ] )
```

Parameters

string-expression

The string to decompress. Binary values can also be passed to this function. This parameter is case sensitive, even in case-insensitive databases.

compression-algorithm-alias

Alias (string) for the algorithm to use for decompression. The supported values are 'zip' and 'gzip' (both are based on the same algorithm, but use different headers and trailers).

Zip is a widely supported compression algorithm. Gzip is compatible with the gzip utility on UNIX and Linux, whereas the zip algorithm is not.

If no algorithm is specified, the function attempts to detect which algorithm was used to compress the string. If the incorrect algorithm is specified, or the correct algorithm cannot be detected, the string is not decompressed.

Returns

LONG BINARY

Remarks

This function can be used to decompress a value that was compressed using the COMPRESS function.

You do not need to use the DECOMPRESS function on values that are stored in a compressed column. Compression and decompression of values in a compressed column are handled automatically by the database server.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example uses the DECOMPRESS function to decompress values from the Attachment column of a fictitious table, TableA:

```
SELECT DECOMPRESS ( Attachment, 'gzip' )
FROM TableA;
```

Since DECOMPRESS returns binary values, if the original values were of a character type, such as LONG VARCHAR, a CAST can be applied to return human-readable values:

```
SELECT CAST ( DECOMPRESS ( Attachment, 'gzip' )
AS LONG VARCHAR ) FROM TableA;
```

Related Information

Column Compression Considerations
String Functions [page 222]
COMPRESS Function [String] [page 286]

1.3.2.65 DECRYPT Function [String]

Decrypts the string using the supplied key and returns a LONG BINARY value.

Syntax

```
DECRYPT( string-expression , key [ , algorithm-format [ , initialization-
vector ] ] )
```

```
algorithm-format :
algorithm [ ( format ) ]
```

```
algorithm :
AES
| AES256
| AES256CTR
| AES_FIPS
| AES256_FIPS
| AES256CTR_FIPS
```

```
| ARIA256
| ARIA256CTR
| RSA
| RSA_FIPS
```

```
format :
FORMAT= { RAW[; padding ] | INTERNAL }
```

```
padding :
PADDING= { PKCS5
| ZEROES
| OAEP
| PKCS1
| ALL
| NONE }
```

Parameters

string-expression

The string to be decrypted. Binary values are supported. This parameter is case sensitive, even in case-insensitive databases.

key

The encryption key (string) that is required to decrypt the `string-expression`. For AES, this value must be the same encryption key that was used to encrypt the `string-expression` to obtain the original value that was encrypted. This parameter is case sensitive, even in case-insensitive databases.

Specify keys in PEM format for RSA.

> ⚠ Caution
>
> For strongly encrypted databases, store a copy of the key in a safe location. If you lose the encryption key, there is no way to access the data, even with the assistance of Technical Support. The database must be discarded and you must create a new database.

algorithm-format

This optional string parameter specifies the type of algorithm, format, and padding that was used to encrypt the `string-expression`.

algorithm

This optional string parameter specifies the type of algorithm originally used to encrypt the `string-expression`. Specify one of the following formats:

AES

The data is encrypted using the AES algorithm.

If `algorithm-format` is not specified, then AES is used by default.

For the AES algorithm, `padding` can be PKCS5, ZEROES, or NONE. The default padding is PKCS5.

AES256 The data is encrypted using the AES 256-bit algorithm. For AES256, `padding` can be PKCS5, ZEROES, and NONE (if FORMAT=RAW).

AES256CTR

The data is encrypted using the AES 256-bit algorithm with CTR block cipher mode. CTR is stronger than CBC.

AES_FIPS

The data is encrypted using the FIPS-certified version of the AES algorithm.

If the database server was started using the -fips server option, AES_FIPS is used as the default. For AES_FIPS, `padding` can be PKCS5, ZEROES, and NONE (if FORMAT=RAW).

AES256_FIPS The data is encrypted using the FIPS-certified version of the AES 256-bit algorithm. For AES256_FIPS, `padding` can be PKCS5, ZEROES, and NONE (if FORMAT=RAW).

AES256CTR_FIPS The data is encrypted using the FIPS-certified version of the AES 256-bit algorithm with CTR block cipher mode.

ARIA256

The data is encrypted using the ARIA 256-bit algorithm with CBC block cipher mode. There is no FIPS variant for this algorithm.

ARIA256CTR

The data is encrypted using the ARIA 256-bit algorithm with CTR block cipher mode. CTR is stronger than CBC. There is no FIPS variant for this algorithm.

RSA

For the RSA algorithm, when encrypting with a public key, `padding` can be PKCS1, OAEP, or NONE. When encrypting with a private key, `padding` must be PKCS1. The default padding is PKCS1.

If the RSA algorithm is specified, then the `initialization-vector` parameter is ignored and FORMAT=RAW is ignored.

If a public key encrypts the message, then a private key must decrypt it. Using the same key for encryption and decryption fails unless PADDING=NONE. However, if PADDING=NONE is set and the incorrect key is supplied, then the function succeeds but returns meaningless data.

> **i** Note
>
> The maximum message length for RSA encryption is equal to the key size minus 11 bytes for PKCS1 padding and the key size minus 42 bytes for OAEP padding. If you specify PADDING=NONE, then the message must be equal to the key size. Unlike AES, the length of the output is not the same as the length of the input when using RSA encryption.

RSA_FIPS The same as RSA except that the data is encrypted using the FIPS-certified version of the RSA algorithm.

FORMAT clause

Use the optional FORMAT clause to specify the storage format for the data. If the data was stored in the proprietary storage format, then specify INTERNAL . If the encrypted data was stored as-is (that is, it can be decrypted by any software that can decrypt the specified algorithm), then specify RAW. For data stored as RAW, specify the `initialization-vector` parameter.

PADDING clause

Use the optional PADDING clause to specify the padding type for AES and RSA encryption. For AES encryption, you must also specify FORMAT=RAW.

The padding type for decryption must match that used for encryption unless PADDING=ALL is used.

PKCS5

The data is padded by using the PKCS#5 algorithm. The encrypted data is 1-16 bytes longer than the decrypted data. This option is only available for AES encryption.This is the default padding for AES encryption.

ZEROES

The data is padded with zeros (0) before encryption. The encrypted data is 0-15 bytes longer than the decrypted data. When the encrypted data is decrypted, the result is also padded with zeros.

OAEP The data is padded using Optimal Asymmetric Encryption Padding. This option is only available for RSA encryption (RSA or RSA_FIPS).

PKCS1 The data is padded using the PKCS#1 algorithm. This option is only available for RSA encryption (RSA or RSA_FIPS). This option is the default for RSA encryption (RSA or RSA_FIPS).

NONE

The data is not padded. The input data must be a multiple of the cipher block length (16-bytes) for AES, or exactly equal to the key size for RSA.

ALL

Each valid padding type is attempted to be used until one of them works.

initialization-vector

Specify `initialization-vector` when `format` is set to RAW. The string cannot be longer than 16 bytes. Any value less than 16 bytes is padded with 0 bytes. This string cannot be set to NULL. `initialization-vector` is ignored when `format` is set to INTERNAL

Returns

LONG BINARY

Remarks

The DECRYPT function decrypts a `string-expression` that was encrypted with the ENCRYPT function. This function returns a LONG BINARY value with the same number of bytes as the input string, unless the data is in raw format. When FORMAT=RAW, the length of the returned value depends on the padding format.

For AES, to successfully decrypt a `string-expression`, use the same encryption key that was used to encrypt the data. When FORMAT=RAW, use the same initialization vector and padding format that was used to encrypt the data. Data in raw format can be decrypted outside of the database server.

For RSA, if you specify an incorrect encryption key, then an error is generated unless FORMAT=RAW is specified. When you specify FORMAT=RAW and an incorrect encryption key or an incorrect initialization vector, the decryption fails silently.

> ⚠ Caution
>
> For strongly encrypted data, store a copy of the key in a safe location. If you lose the encryption key, then there is no way to access the data, even with the assistance of Technical Support.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example decrypts a user's password from the user_info table. The CAST function is used to convert the password back to a CHAR data type because the DECRYPT function converts values to the LONG BINARY data type, which is unreadable.

```
SELECT CAST( DECRYPT( user_pwd, '8U3dkA' ) AS CHAR(100) ) FROM user_info;
```

The following example updates the `secret` column with an encrypted version of the `password` column. The data is encrypted using encryption key 'TheEncryptionKey', raw-format AES encryption, and the initialization vector 'ThisIsTheIV'. Default PKCS#5 padding is used.

```
CREATE OR REPLACE TABLE SensitiveData
(
    username char(30), password char(30), secret binary(48)
);
INSERT INTO SensitiveData (username, password)
VALUES
    ('Martin',  'topXsecret1'),
    ('Jasmine', 'my_big_secret'),
    ('Aidan',   'Shortcutsmakelongdelays');
UPDATE SensitiveData
    SET secret = ENCRYPT( password, 'TheEncryptionKey', 'AES(FORMAT=RAW)',
'ThisIsTheIV' );
```

The encrypted text in the `secret` column is decrypted using the DECRYPT function.

```
SELECT
    username,
    password,
    CAST(DECRYPT( secret, 'TheEncryptionKey','AES(FORMAT=RAW;PADDING=PKCS5)',
'ThisIsTheIV' ) AS LONG VARCHAR)
    AS revealed
FROM SensitiveData;
```

Related Information

Raw Encryption
How to Encrypt Tables, Columns, and Materialized Views
String Functions [page 222]
Encrypting and Decrypting Messages
ENCRYPT Function [String] [page 350]
ISENCRYPTED Function [System] [page 427]
-fips Database Server Option

1.3.2.66 DEGREES Function [Numeric]

Converts a number from radians to degrees.

Syntax

```
DEGREES( numeric-expression )
```

Parameters

numeric-expression

An angle in radians.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns the degrees of the angle given by `numeric-expression`. If the parameter is NULL, the result is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 29.79380534680281:

```
SELECT DEGREES( 0.52 );
```

1.3.2.67 DENSE_RANK Function [Ranking]

Calculates the rank of a value in a partition. For tied values, the DENSE_RANK function does not leave gaps in the ranking sequence.

Syntax

```
DENSE_RANK( ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Returns

INTEGER

Remarks

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause. More information is available in the `window-spec` definition of the WINDOW clause.

Standards

ANSI/ISO SQL Standard

The DENSE_RANK function comprises part of optional ANSI/ISO SQL Language Feature T612, "Advanced OLAP operations".

SQL Language Feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references, is supported.

Optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions" is not supported. The software does not support the arguments of an aggregate function to include both a column reference from the query block containing the DENSE_RANK function, combined with an outer reference.

Example

The following example returns a result set that provides a ranking of the employees' salaries in Utah and New York. Although 19 records are returned in the result set, only 18 rankings are listed because of a 7th-place tie between the 7th and 8th employee in the list, who have identical salaries. Instead of ranking the 9th employee as '9', the employee is listed as '8' because the DENSE_RANK function does not leave gaps in the ranks.

```
SELECT DepartmentID, Surname, Salary, State,
DENSE_RANK() OVER (ORDER BY Salary DESC) AS SalaryRank
FROM GROUPO.Employees
WHERE State IN ('NY','UT');
```

Here is the result set:

DepartmentID	Surname	Salary	State	SalaryRank
100	Shishov	72995.000	UT	1
100	Wang	68400.000	UT	2
100	Cobb	62000.000	UT	3
400	Morris	61300.000	UT	4
300	Davidson	57090.000	NY	5
200	Martel	55700.000	NY	6
400	Blaikie	54900.000	NY	7
100	Diaz	54900.000	UT	7
100	Driscoll	48023.000	UT	8
400	Hildebrand	45829.000	UT	9
100	Whitney	45700.000	NY	10
100	Guevara	42998.000	NY	11
100	Soo	39075.000	NY	12
200	Goggin	37900.000	UT	13
400	Wetherby	35745.000	NY	14
400	Ahmed	34992.000	NY	15
500	Rebeiro	34576.000	UT	16

DepartmentID	Surname	Salary	State	SalaryRank
300	Bigelow	31200.000	UT	17
500	Lynch	24903.000	UT	18

Related Information

Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
CUME_DIST Function [Ranking] [page 311]
PERCENT_RANK Function [Ranking] [page 485]
RANK Function [Ranking] [page 501]

1.3.2.68 DIFFERENCE Function [String]

Returns the difference in the SOUNDEX values between the two string expressions.

Syntax

```
DIFFERENCE ( string-expression-1 , string-expression-2 )
```

Parameters

string-expression-1

The first SOUNDEX argument.

string-expression-2

The second SOUNDEX argument.

Returns

SMALLINT

Remarks

The DIFFERENCE function compares the SOUNDEX values of two strings and evaluates the similarity between them, returning a value from 0 through 4, where 4 is the best match.

This function always returns some value. The result is NULL only if one of the arguments are NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns similarity between the words test and chest:

```
SELECT DIFFERENCE( 'test', 'chest' );
```

Related Information

String Functions [page 222]
SOUNDEX Function [String] [page 559]

1.3.2.69 DOW Function [Date and Time]

Returns a number from 1 to 7 representing the day of the week of a date, where Sunday=1, Monday=2, and so on.

Syntax

```
DOW( date-expression )
```

Parameters

date-expression

The value (of type DATE) to be evaluated.

Returns

SMALLINT

Remarks

The DOW function is not affected by the value specified for the first_day_of_week database option. For example, even if first_day_of_week is set to Monday, the DOW function returns a 2 for Monday.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 5:

```
SELECT DOW( '1998-07-09' );
```

The following statement returns the value 1:

```
SELECT DOW( CAST( '2010/05/30 11:33:00.000000+04:00' as TIMESTAMP WITH TIME
ZONE ));
```

The following statement queries the Employees table and returns the employee StartDate, expressed as the number of the day of the week:

```
SELECT DOW( StartDate ) FROM GROUPO.Employees;
```

1.3.2.70 ENCRYPT Function [String]

Encrypts the specified value using the supplied encryption key and returns a LONG BINARY value.

Syntax

```
ENCRYPT( string-expression , key [ , algorithm-format [ , initialization-
vector ] ] )
```

```
algorithm-format :
algorithm [ ( format ) ]
```

```
algorithm :
AES
| AES256
| AES256CTR
| AES256CTR_FIPS
| AES_FIPS
| AES256_FIPS
| ARIA256
| ARIA256CTR
| RSA
| RSA_FIPS
```

```
format :
FORMAT= { RAW[; padding ] | INTERNAL }
```

```
padding :
PADDING= { PKCS5
| ZEROES
| OAEP
| PKCS1
| ALL
| NONE }
```

Parameters

string-expression

The string to be encrypted. Binary values are supported. This parameter is case sensitive, even in case-insensitive databases.

key

The encryption key (string) that is required to decrypt the `string-expression`. For AES, this value must be the same encryption key that was used to encrypt the `string-expression` to obtain the original value that was encrypted. This parameter is case sensitive, even in case-insensitive databases.

Specify keys in PEM format for RSA.

> ⚠ Caution
>
> For strongly encrypted databases, store a copy of the key in a safe location. If you lose the encryption key, there is no way to access the data, even with the assistance of Technical Support. The database must be discarded and you must create a new database.

algorithm-format

This optional string parameter specifies the type of algorithm, format, and padding to use when encrypting the `string-expression`.

algorithm

This optional string parameter specifies the type of algorithm used to encrypt the `string-expression`. Specify one of the following formats:

AES

The data is encrypted using the AES 128-bit algorithm.

If `algorithm-format` is not specified, then AES is used by default.

For the AES algorithm, `padding` can be PKCS5, ZEROES, or NONE. The default padding is PKCS5.

AES256

The data is encrypted using the AES 256-bit algorithm with CBC block cipher mode. For AES256, `padding` can be PKCS5, ZEROES, and NONE (if FORMAT=RAW).

AES256CTR

The data is encrypted using the AES 256-bit algorithm with CTR block cipher mode. CTR is stronger than CBC.

AES256CTR_FIPS

The data is encrypted using the FIPS-certified version of the AES 256-bit algorithm with CTR block cipher mode. CTR is stronger than CBC.

AES_FIPS

The data is encrypted using the FIPS-certified version of the AES 128-bit algorithm.

If the database server was started using the -fips server option, AES_FIPS is used as the default. For AES_FIPS, `padding` can be PKCS5, ZEROES, and NONE (if FORMAT=RAW).

AES256_FIPS The data is encrypted using the FIPS-certified version of the AES 256-bit algorithm. For AES256_FIPS, `padding` can be PKCS5, ZEROES, and NONE (if FORMAT=RAW).

ARIA256

The data is encrypted using the ARIA 256-bit algorithm with CBC block cipher mode. There is no FIPS variant for this algorithm.

ARIA256CTR

The data is encrypted using the ARIA 256-bit algorithm with CTR block cipher mode. CTR is stronger than CBC. There is no FIPS variant for this algorithm.

RSA

For the RSA algorithm, when encrypting with a public key, `padding` can be PKCS1, OAEP, or NONE. When encrypting with a private key, `padding` must be PKCS1. The default padding is PKCS1.

If the RSA algorithm is specified, then the `initialization-vector` parameter is ignored and FORMAT=RAW is ignored.

If a public key encrypts the message, then a private key must decrypt it. Using the same key for encryption and decryption fails unless PADDING=NONE. However, if PADDING=NONE is set and the incorrect key is supplied, then the function succeeds but returns meaningless data.

> **i** Note
>
> The maximum message length for RSA encryption is equal to the key size minus 11 bytes for PKCS1 padding and the key size minus 42 bytes for OAEP padding. If you specify PADDING=NONE, then the message must be equal to the key size. Unlike AES, the length of the output is not the same as the length of the input when using RSA encryption.

RSA_FIPS The same as RSA except that the data is encrypted using the FIPS-certified version of the RSA algorithm.

FORMAT clause

Use the optional FORMAT clause to specify the storage format for the data. If the data was stored in the proprietary storage format, then specify INTERNAL . If the encrypted data was stored as-is (that is, it can be decrypted by any software that can decrypt the specified algorithm), then specify RAW. For data stored as RAW, specify the `initialization-vector` parameter.

PADDING clause

Use the optional PADDING clause to specify the padding type for AES and RSA encryption. For AES encryption, you must also specify FORMAT=RAW.

The padding type for decryption must match that used for encryption unless PADDING=ALL is used.

PKCS5

The data is padded by using the PKCS#5 algorithm. The encrypted data is 1-16 bytes longer than the decrypted data. This option is only available for AES encryption.This is the default padding for AES encryption.

ZEROES

The data is padded with zeros (0) before encryption. The encrypted data is 0-15 bytes longer than the decrypted data. When the encrypted data is decrypted, the result is also padded with zeros.

OAEP The data is padded using Optimal Asymmetric Encryption Padding. This option is only available for RSA encryption (RSA or RSA_FIPS).

PKCS1 The data is padded using the PKCS#1 algorithm. This option is only available for RSA encryption (RSA or RSA_FIPS). This option is the default for RSA encryption (RSA or RSA_FIPS).

NONE

The data is not padded. The input data must be a multiple of the cipher block length (16-bytes) for AES, or exactly equal to the key size for RSA.

initialization-vector

Specify `initialization-vector` when `format` is set to RAW. The string cannot be longer than 16 bytes. Any value less than 16 bytes is padded with 0 bytes. This string cannot be set to NULL. `initialization-vector` is ignored when `format` is set to INTERNAL

Returns

LONG BINARY

Remarks

The LONG BINARY value returned by this function is up to 31 bytes longer than the input `string-expression`. The value returned by this function is not human-readable. Use the DECRYPT function to decrypt a `string-expression` that was encrypted with the ENCRYPT function. For AES, to successfully decrypt a `string-expression`, use the same encryption key and algorithm that were used to encrypt the data. If you specify an incorrect encryption key, then an error is generated. A lost key results in inaccessible data, from which there is no recovery.

If you are storing encrypted values in a table, then the column should be BINARY or LONG BINARY so that character set conversion is not performed on the data.

When FORMAT=RAW is specified, the data is encrypted using raw encryption. Specify the encryption key, initialization vector, and, optionally, the padding format. These same values must be specified when decrypting the data. The decryption can be performed outside of the database server or by using the DECRYPT function.

Do not use raw encryption when the data is to be encrypted and decrypted only within the database server because you must specify the initialization vector and the padding, and the encryption key cannot be verified during decryption.

> **i** Note
>
> For the ISENCRYPTED function to return meaningful results, data must be encrypted using the ENCRYPT function with AES/AES256 and must not use FORMAT=RAW.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following trigger encrypts the user_pwd column of the user_info table. This column contains users' passwords, and the trigger fires whenever a password value is changed.

```
CREATE TRIGGER encrypt_updated_pwd
BEFORE UPDATE OF user_pwd
ON user_info
REFERENCING NEW AS new_pwd
FOR EACH ROW
BEGIN
    SET new_pwd.user_pwd=ENCRYPT( new_pwd.user_pwd, '8U3dkA' );
END;
```

The following example updates the `secret` column with an encrypted version of the `password` column. The data is encrypted using encryption key 'TheEncryptionKey', raw-format AES encryption, PKCS#5 padding (the default), and the initialization vector 'ThisIsTheIV'.

```
CREATE OR REPLACE TABLE SensitiveData
(
    username char(30), password char(30), secret binary(48)
);
INSERT INTO SensitiveData (username, password)
VALUES
    ('Martin',  'topXsecret1'),
    ('Jasmine', 'my_big_secret'),
    ('Aidan',   'Shortcutsmakelongdelays');
UPDATE SensitiveData
    SET secret = ENCRYPT( password, 'TheEncryptionKey',
'AES(FORMAT=RAW;PADDING=PKCS5)', 'ThisIsTheIV' );
```

```
SELECT *, LENGTH(secret) FROM SensitiveData;
```

Related Information

How to Encrypt Tables, Columns, and Materialized Views
Encrypting and Decrypting Messages
DECRYPT Function [String] [page 340]
ISENCRYPTED Function [System] [page 427]
-fips Database Server Option

1.3.2.71 ERROR_LINE Function [Miscellaneous]

Returns the line number of the procedure or batch where the error occurred that invoked the CATCH block of a TRY...CATCH statement.

Syntax

```
ERROR_LINE( )
```

Returns

UNSIGNED INTEGER representing the line number within the stored procedure or the compound statement where an error occurred.

Remarks

Call this function anywhere within a CATCH block. This function reports information about the current error when it is invoked within an error handler, a nested compound statement, a function, or a procedure.

This function returns line numbers as found in the proc_defn column of the SYSPROCEDURE system table for the procedure. These line numbers might differ from those of the source definition used to create the procedure.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

When executed within a handler that was invoked by a division by zero error on line 15 of the procedure u1.proc1, the following statement `SELECT ERROR_LINE( ), ERROR_MESSAGE( ), ERROR_PROCEDURE( )` returns a result similar to the following one:

```
15, 'Division by zero',  '"u1"."proc1"'
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
Example: Creating an Error Logging Procedure That Can be Called by an Exception Handler
SYSPROCEDURE System View [page 1931]
SYSPROCEDURE System View [page 1931]

1.3.2.72 ERROR_MESSAGE Function [Miscellaneous]

Returns the message text of the error that invoked the CATCH block of a TRY...CATCH statement.

Syntax

ERROR_MESSAGE( )

Returns

VARCHAR containing the error message of the error that invoked the CATCH block.

Remarks

Call this function anywhere within a CATCH block. This function returns the active error message anywhere in the error handler, while the ERRORMSG function, when called with no parameters, only returns the error message when invoked in the first statement of the error handler.

The parameters in the error message are replaced with actual values.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

When executed within a handler that was invoked by a division by zero error on line 15 of the procedure u1.proc1, the following statement `SELECT ERROR_LINE(), ERROR_MESSAGE(), ERROR_PROCEDURE()` returns the following result:

```
15, 'Division by zero',  '"u1"."proc1"'
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
Example: Creating an Error Logging Procedure That Can be Called by an Exception Handler

1.3.2.73 ERROR_PROCEDURE Function [Miscellaneous]

Returns the name of the procedure within which the error that caused the exception handler to run occurred.

Syntax

```
ERROR_PROCEDURE( )
```

Returns

VARCHAR containing the qualified name of the procedure where the exception has occurred. If the compound statement is not part of a procedure, function, trigger, or event, the type of batch (`<watcom_batch>` or `<tsql_batch>`) is returned instead of the procedure owner and name.

Remarks

ERROR_PROCEDURE can be called anywhere within an exception handler.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following batch illustrates division by zero exception handling.

```
BEGIN
   DECLARE divTest INT;
   SET divTest = 1 / 0;
   SELECT 'No error';
   EXCEPTION WHEN OTHERS THEN
      SELECT 'Exception: SQLCODE = ' || ERROR_SQLCODE() ||
                     ', SQLSTATE = ' || ERROR_SQLSTATE() ||
                     ', PROCEDURE = ' || ERROR_PROCEDURE();
END;
```

The result of executing this batch is:

```
Exception: SQLCODE = -628, SQLSTATE = 22012, PROCEDURE = <watcom_batch>
```

The following procedure also illustrates division by zero exception handling.

```
CREATE OR REPLACE PROCEDURE ExceptionDemo()
BEGIN
   DECLARE divTest INT;
   SET divTest = 1 / 0;
   SELECT 'No error';
   EXCEPTION WHEN OTHERS THEN
      SELECT 'Exception: SQLCODE = ' || ERROR_SQLCODE() ||
                     ', SQLSTATE = ' || ERROR_SQLSTATE() ||
                     ', PROCEDURE = ' || ERROR_PROCEDURE();
END;
CALL ExceptionDemo()
```

The result of executing this procedure is:

```
Exception: SQLCODE = -628, SQLSTATE = 22012, PROCEDURE = "DBA"."ExceptionDemo"
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
Example: Creating an Error Logging Procedure That Can be Called by an Exception Handler

1.3.2.74 ERROR_SQLCODE Function [Miscellaneous]

Returns the SQLCODE of the error that invoked the error handler.

Syntax

ERROR_SQLCODE( )

Returns

SIGNED INTEGER with the value of the SQLCODE of the error that invoked the error handler.

Remarks

This function can be called anywhere within an error handler.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following batch illustrates division by zero exception handling.

```
BEGIN
   DECLARE divTest INT;
   SET divTest = 1 / 0;
   SELECT 'No error';
   EXCEPTION WHEN OTHERS THEN
      SELECT 'Exception: SQLCODE = ' || ERROR_SQLCODE() ||
                     ', SQLSTATE = ' || ERROR_SQLSTATE() ||
                     ', PROCEDURE = ' || ERROR_PROCEDURE();
END;
```

The result of executing this batch is:

```
Exception: SQLCODE = -628, SQLSTATE = 22012, PROCEDURE = <watcom_batch>
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
Example: Creating an Error Logging Procedure That Can be Called by an Exception Handler

1.3.2.75 ERROR_SQLSTATE Function [Miscellaneous]

Returns the SQLSTATE of the error that invoked the error handler.

Syntax

```
ERROR_SQLSTATE( )
```

Returns

CHAR(5) representing the SQLSTATE of the error that invoked the error handler.

Remarks

This function can be called anywhere within an error handler.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following batch illustrates division by zero exception handling.

```
BEGIN
   DECLARE divTest INT;
   SET divTest = 1 / 0;
   SELECT 'No error';
   EXCEPTION WHEN OTHERS THEN
      SELECT 'Exception: SQLCODE = ' || ERROR_SQLCODE() ||
                     ', SQLSTATE = ' || ERROR_SQLSTATE() ||
                     ', PROCEDURE = ' || ERROR_PROCEDURE();
END;
```

The result of executing this batch is:

```
Exception: SQLCODE = -628, SQLSTATE = 22012, PROCEDURE = <watcom_batch>
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
Example: Creating an Error Logging Procedure That Can be Called by an Exception Handler

1.3.2.76 ERROR_STACK_TRACE Function [Miscellaneous]

Returns a calling sequence stack trace for the error that invoked the error handler.

Syntax

```
ERROR_STACK_TRACE( )
```

Returns

LONG VARCHAR representing the stack trace of the error that invoked the error handler. If the compound statement is not part of a procedure, function, trigger, or event, the type of batch (`<watcom_batch>` or `<tsql_batch>`) is returned instead of the procedure name.

Remarks

The result contains lines of text delimited by line feed (\n) characters. Each line of the returned value contains the qualified procedure name or batch type (if any) of the statement on the stack, followed by the line number of the statement. The last line of the returned value is not terminated by a line feed character.

This function returns line numbers as found in the proc_defn column of the SYSPROCEDURE system table for the procedure. These line numbers might differ from those of the source definition used to create the procedure.

This function returns the same information as the sa_error_stack_trace system procedure.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following set of procedures (with line numbers added for illustration) can be used to obtain the error stack:

```
1 CREATE OR REPLACE PROCEDURE proc1()
2 BEGIN TRY
3    CALL proc2();
4 END TRY
5 BEGIN CATCH
6    SELECT * FROM sa_split_list(ERROR_STACK_TRACE(), '\n' );
7 END CATCH;
1 CREATE OR REPLACE PROCEDURE proc2()
2 BEGIN
3    CALL proc3();
4 END;
1 CREATE OR REPLACE PROCEDURE proc3()
2 BEGIN
3    DECLARE v INTEGER = 0;
4    SET v = 1 / v;
5 END;
CALL proc1();
```

This call returns the following result set:

```
line_num   row_value
       1   "DBA"."proc1" : 3
       2   "DBA"."proc2" : 3
       3   "DBA"."proc3" : 4
```

If RESIGNAL is used in the error handler, and the resignaled error is handled, the error stack reported in the second handler contains the stack trace of the original error, the record of the RESIGNAL, and the stack of the resignaled exception. For example:

```
CREATE OR REPLACE PROCEDURE proc1()
BEGIN TRY
    BEGIN TRY
        DECLARE v INTEGER = 0;
        SET v = 1 / v;
    END TRY
    BEGIN CATCH
        CALL proc2();
    END CATCH
END TRY
BEGIN CATCH
    SELECT * FROM sa_split_list(ERROR_STACK_TRACE(), '\n' );
END CATCH;
CREATE OR REPLACE PROCEDURE proc2()
BEGIN
    CALL proc3();
END;
CREATE OR REPLACE PROCEDURE proc3()
BEGIN
    RESIGNAL;
```

```
END;
CALL proc1();
```

This call returns the following result string:

```
line_num    row_value
       1    "DBA"."proc1" : 8
       2    "DBA"."proc2" : 3
       3    RESIGNAL: "DBA"."proc3" : 3
       4    "DBA"."proc1" : 5
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
Example: Creating an Error Logging Procedure That Can be Called by an Exception Handler
SYSPROCEDURE System View [page 1931]

1.3.2.77 ERRORMSG Function [Miscellaneous]

Provides the error message for the current error, or for a specified SQLSTATE or SQLCODE value.

Syntax

```
ERRORMSG( [ sqlstate | sqlcode ] )
```

```
sqlstate: string
```

```
sqlcode: integer
```

Parameters

sqlstate

The SQLSTATE value for which the error message is to be returned.

sqlcode

The SQLCODE value for which the error message is to be returned.

Returns

VARCHAR containing the error message.

Remarks

If no argument is supplied, the error message for the current state is supplied. Any substitutions (such as table names and column names) are made.

If an argument is supplied, the error message for the supplied SQLSTATE or SQLCODE is returned, with no substitutions. Table names and column names are supplied as placeholders (%1).

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the error message for SQLCODE -813:

```
SELECT ERRORMSG( -813 );
```

Related Information

SQL Anywhere Error Messages Sorted by SQLSTATE
SQL Anywhere Error Messages Sorted by SQLCODE

1.3.2.78 ESTIMATE Function [Miscellaneous]

Returns selectivity estimates as a percentage calculated by the query optimizer, based on specified parameters.

Syntax

```
ESTIMATE( column-name [ , value [ , relation-string ] ] )
```

Parameters

column-name

The column used in the estimate.

value

The value to which the column is compared. The default is NULL.

relation-string

The comparison operator used for the comparison, enclosed in single quotes. Possible values for this parameter are: '=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' , and '!>'. The default is '='.

Returns

REAL

Remarks

This function returns selectivity estimates for the predicate `column-name relation-string value`. If `value` is NULL and the relation string is '=', the selectivity is for the predicate `column-name IS NULL`. If `value` is NULL and the relation string is '!=' or '<>', the selectivity is for the predicate `column-name IS NOT NULL`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the percentage of EmployeeID values estimated to be greater than 200. The precise value depends on the actions you have carried out on the database.

```
SELECT FIRST ESTIMATE( EmployeeID, 200, '>' )
   FROM GROUPO.Employees
   ORDER BY 1;
```

Related Information

Selectivity Estimate Sources
Selectivity Information in the Graphical Plan
INDEX_ESTIMATE Function [Miscellaneous] [page 421]
ESTIMATE_SOURCE Function [Miscellaneous] [page 367]
EXPERIENCE_ESTIMATE Function [Miscellaneous] [page 376]

1.3.2.79 ESTIMATE_SOURCE Function [Miscellaneous]

Provides the source for selectivity estimates used by the query optimizer.

Syntax

```
ESTIMATE_SOURCE( column-name [ , value [ , relation-string ] ] )
```

Parameters

column-name

The column used in the estimate.

value

The value to which the column is compared. The default is NULL.

relation-string

The comparison operator used for the comparison, enclosed in single quotes. Possible values for this parameter are: '=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' , and '!>'. The default is '='.

Returns

The following list shows the selectivity estimate sources that ESTIMATE_SOURCE returns.

Value	Selectivity estimate source
Statistics	Stored column statistics
Column	Average of all values stored in the column statistics
Index	Index probes
Guess	Built-in guesses that are defined for each type of predicate. This is returned only when there is no relevant index to use, no statistics have been collected for the referenced columns, or the predicate is a complex predicate.
Computed	Other sources than the ones described above
Always	Returned when the specified predicate is always true
Combined	One or more of the above sources
Bounded	Returned when there are upper and/or lower bounds placed on the selectivity estimate

Remarks

This function returns the source of the selectivity estimate for the predicate `column-name relation-string value`. If `value` is NULL and the relation string is '=', the selectivity source is for the predicate `column-name IS NULL`. If `value` is NULL and the relation string is '!=' or '<>', the selectivity source is for the predicate `column-name IS NOT NULL`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the selectivity source Index for evaluating whether the first value in the EmployeeID column is greater than 200. Returning Index means that the query optimizer used an index to estimate the selectivity.

```
SELECT FIRST ESTIMATE_SOURCE( EmployeeID, 200, '>' )
   FROM GROUPO.Employees
   ORDER BY 1;
```

Related Information

Selectivity Estimate Sources
ESTIMATE Function [Miscellaneous] [page 366]
INDEX_ESTIMATE Function [Miscellaneous] [page 421]

1.3.2.80 EVENT_CONDITION Function [System]

Specifies when an event handler is triggered.

Syntax

```
EVENT_CONDITION( condition-name )
```

Parameters

condition-name

The condition triggering the event. The possible values are preset in the database, and are case insensitive. Each condition is valid only for certain event types. The conditions and the events for which they are valid are as follows:

Condition name	Units	Valid for...	Comments
DBFreePercent	n/a	DBDiskSpace	The percentage of free disk space remaining for the database
DBFreeSpace	MB	DBDiskSpace	The free disk space available for the database in megabytes
DBSize	MB	GrowDB	The size of the database in megabytes
ErrorNumber	n/a	RAISERROR	The error code number
IdleTime	seconds	ServerIdle	The time that the server is idle in seconds
Interval	seconds	All	Time since handler last executed
LogFreePercent	n/a	LogDiskSpace	The percentage of free disk space remaining for the log.
LogFreeSpace	MB	LogDiskSpace	The free disk space remaining for the log in megabytes.

Condition name	Units	Valid for...	Comments
LogSize	MB	GrowLog	The size of the log in megabytes
RemainingValues	integer	GlobalAutoincrement	The number of remaining values
TempFreePercent	n/a	TempDiskSpace	The percentage of free disk space in the temporary file space
TempFreeSpace	MB	TempDiskSpace	The free disk space available for the temporary file space in megabytes
TempSize	MB	GrowTemp	The size of the temporary file space in megabytes

Returns

INT

Remarks

The EVENT_CONDITION function returns NULL when not called from an event.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following event definition uses the EVENT_CONDITION function:

```
CREATE EVENT LogNotifier
TYPE LogDiskSpace
WHERE event_condition( 'LogFreePercent' ) < 50
HANDLER
BEGIN
   MESSAGE 'LogNotifier message'
END;
```

Related Information

CREATE EVENT Statement [page 847]

1.3.2.81 EVENT_CONDITION_NAME Function [System]

Lists the possible parameters for EVENT_CONDITION.

Syntax

```
EVENT_CONDITION_NAME( integer )
```

Parameters

integer

Must be greater than or equal to zero.

Returns

VARCHAR

Remarks

You can use the EVENT_CONDITION_NAME function to obtain a list of all arguments for the EVENT_CONDITION function by looping over integers until the function returns NULL.

The EVENT_CONDITION_NAME function returns NULL when not called from an event.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

CREATE EVENT Statement [page 847]

1.3.2.82 EVENT_PARAMETER Function [System]

Provides context information for event handlers.

Syntax

```
EVENT_PARAMETER( context-name )
```

```
context-name :
AppInfo
| ConnectionID
| DisconnectReason
| EventName
| Executions
| MirrorServerName
| NumActive
| ScheduleName
| SQLCODE
| TableName
| User
| condition-name
```

Parameters

context-name

One of the preset strings. The strings must be quoted, are case insensitive, and carry the following information:

AppInfo

The value of the AppInfo connection property for the connection that caused the event to be triggered. Use the following statement to see the value of the property outside the context of the event:

```
SELECT CONNECTION_PROPERTY( 'AppInfo' );
```

The AppInfo string contains the computer name and application name of the client connection for Embedded SQL, ODBC, OLE DB, ADO.NET, and SQL Anywhere JDBC driver connections.

ConnectionID

The connection ID of the connection that caused the event to be triggered.

DisconnectReason

A string indicating the reason the connect was terminated. This parameter is valid only for Disconnect events. Possible results include:

abnormal

A disconnect occurred as a result of the client application terminating abnormally before disconnecting from the database, or as a result of a communication failure between the client and server computers.

connect failed

A connection attempt failed.

drop connection

A DROP CONNECTION statement was executed.

from client

The client application disconnected.

inactive

No requests were received for the period specified by the -ti server option.

liveness

No liveness packets were received for the period specified by the -tl server option.

EventName

The name of the event that has been triggered.

Executions

The number of times the event handler has been executed.

MirrorServerName

The name of the mirror or arbiter server that lost its connection to the primary server in a database mirroring system.

NumActive

The number of active instances of an event handler. This is useful for limiting an event handler so that only one instance executes at any given time.

ScheduleName

The name of the schedule which caused an event to be fired. If the event was fired manually using TRIGGER EVENT or as a system event, the result will be an empty string. If the schedule was not assigned a name explicitly when it was created, its name is the name of the event.

SQLCODE

The SQLCODE of the error that occurred during a failed connection. This parameter is valid only for ConnectFailed events.

TableName

The name of the table, for use with RemainingValues.

User

The user ID for the user that caused the event to be triggered.

In addition, you can access any of the valid `condition-name` arguments to the EVENT_CONDITION function from the EVENT_PARAMETER function.

The following table indicates which context-name values are valid for which system event types.

System event types	Context-name value
BackupEnd	AppInfo, ConnectionID, EventName, Executions, NumActive, User
Connect	AppInfo, ConnectionID, EventName, Executions, NumActive, User
ConnectFailed	AppInfo, EventName, Executions, NumActive, SQLCODE, User
"Disconnect"	AppInfo, ConnectionID, EventName, Executions, NumActive, User
GlobalAutoincrement	ConnectionID, EventName, Executions, NumActive, TableName, User
"RAISERROR"	AppInfo, ConnectionID, EventName, Executions, NumActive, User
User events	AppInfo, ConnectionID, EventName, Executions, NumActive, User

Returns

VARCHAR

Remarks

The maximum size of values passed to an event is limited by the maximum page size for the server (-gp server option). Values that are longer are truncated to be less than the maximum page size.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example shows how to pass a string parameter to an event. The event displays the time it was triggered in the database server messages window.

```
CREATE EVENT ev_PassedParameter
```

```
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' || event_parameter( 'time' );
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string(current timestamp ) );
```

The following example uses the => parameter syntax instead:

```
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' ||
event_parameter( 'what_time' );
END;
TRIGGER EVENT ev_PassedParameter( what_time => string( current timestamp ) );
```

Related Information

EVENT_CONDITION Function [System] [page 369]
CREATE EVENT Statement [page 847]
TRIGGER EVENT Statement [page 1441]
-gp Database Server Option

1.3.2.83 EXP Function [Numeric]

Returns the result of the base of natural logarithms e raised to the power of the given argument.

Syntax

```
EXP( numeric-expression )
```

Parameters

numeric-expression

The exponent.

Returns

DOUBLE

Remarks

The EXP function returns the result of raising the base of natural logarithms e by the value specified by `numeric-expression`.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Standards

ANSI/ISO SQL Standard

The EXP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The statement returns the value 3269017.3724721107:

```
SELECT EXP( 15 );
```

1.3.2.84 EXPERIENCE_ESTIMATE Function [Miscellaneous]

Returns selectivity estimates as a percentage calculated by the query optimizer, based on specified parameters.

Syntax

```
EXPERIENCE_ESTIMATE( column-name , value [ , relation-string ] )
```

Parameters

column-name

The name of the column that is being investigated.

value

The value to which the column is compared.

relation-string

The comparison operator used for the comparison. Possible values for this parameter are: '=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' , and '!>'. The default is '='.

Returns

REAL

Remarks

If `value` is NULL then the relation strings = and != are interpreted as the IS NULL and IS NOT NULL conditions, respectively.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns 90.3262405396:

```
SELECT DISTINCT EXPERIENCE_ESTIMATE( EmployeeID, 200, '>' )
FROM GROUPO.Employees;
```

Related Information

ESTIMATE Function [Miscellaneous] [page 366]
INDEX_ESTIMATE Function [Miscellaneous] [page 421]
ESTIMATE_SOURCE Function [Miscellaneous] [page 367]

1.3.2.85 EXPLANATION Function [Miscellaneous]

Returns the optimization strategy of a SQL statement as a plain text string.

Syntax

```
EXPLANATION( string-expression [ , cursor-type [ , update-status ] ] )
```

UltraLite:

```
EXPLANATION( string-expression )
```

Parameters

string-expression

The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE, MERGE, or DELETE statement.

cursor-type

A cursor type, expressed as a string. Possible values are asensitive, insensitive, sensitive, or keyset-driven. If `cursor-type` is not specified, asensitive is used by default.

update-status

A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor:

Value	Description
READ-ONLY	The cursor is read-only.
READ-WRITE (default)	The cursor can be read or written to.
FOR UPDATE	The cursor can be read or written to. This is the same as READ-WRITE.

Returns

LONG VARCHAR

Remarks

The execution plan for the query, returned as a string.

The GRAPHICAL_PLAN function offers significantly greater information about access plans, including system properties that may have affected how the statement was optimized.

This information can help you decide which indexes to add or how to structure your database for better performance.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement passes a SELECT statement as a string parameter and returns the plan for executing the query:

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

The following statement returns a string containing the short form of the text plan for an INSENSITIVE cursor over the query SELECT * FROM Departments WHERE ...':

```
SELECT EXPLANATION( 'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100',
   'insensitive', 'read-only' );
```

Related Information

Advanced: Query Execution Plans
PLAN Function [Miscellaneous] [page 488]
GRAPHICAL_PLAN Function [Miscellaneous] [page 393]

1.3.2.86 EXPRTYPE Function [Miscellaneous]

Returns a string that identifies the data type of an expression.

Syntax

```
EXPRTYPE( string-expression , integer-expression )
```

Parameters

string-expression

A SELECT statement. The expression whose data type is to be queried must appear in the SELECT list. If the string is not a valid SELECT statement, NULL is returned.

integer-expression

The position in the SELECT list of the desired expression. The first item in the SELECT list is numbered 1. If the integer-expression value does not correspond to a SELECT list item, NULL is returned.

Returns

LONG VARCHAR

Remarks

For user-defined domains, EXPRTYPE returns the description of the underlying data type, not the domain name. For example, suppose you create a domain, mydomain, and define a table column using mydomain, as follows:

```
CREATE DOMAIN mydomain CHAR(20);
CREATE TABLE mytable( colA mydomain, colB DATETIME );
```

When you execute `SELECT EXPRTYPE( 'SELECT * FROM mytable', 1 )`, the data type returned is char(20), not mydomain.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns `smallint` when executed against the SQL Anywhere sample database:

```
SELECT EXPRTYPE( 'SELECT LineID FROM SalesOrderItems', 1 );
```

Related Information

SQL Data Types [page 129]
sa_describe_query System Procedure [page 1576]

1.3.2.87 EXTENDED_PROPERTY Function [System]

Returns the value of the given database server property. Allows an optional property-specific string parameter to be specified.

Syntax

```
EXTENDED_PROPERTY( { property-id | property-name } [ , property-specific-argument ] )
```

Parameters

property-id

An integer that is the property number of the database server property. This number can be determined from the PROPERTY_NUMBER function. The `property-id` is commonly used when looping through a set of properties.

property-name

The database server property name to query.

HasSecureFeatureKey

Specify a list of features to determine whether the database server has a secured feature key that unlocks all of the features in the list. Returns NULL if `property-specific-argument` is NULL; otherwise returns Yes or No.

HasSecuredFeature

Specify a list of features to determine whether any of the specified features is secured at the global server level. Returns NULL if `property-specific-argument` is NULL; otherwise returns Yes or No.

property-specific-argument

The following database server properties allow you to specify additional arguments, as noted below, to return specific information about the property.

HasSecureFeatureKey `feature-list`

Specify a list of features to determine whether there is a secure feature key that unlocks all of the features in `feature-list`.

HasSecuredFeature `feature-list`

Specify a list of features to determine whether at least one of these features is secured.

Returns

VARCHAR, LONG VARCHAR

Remarks

The EXTENDED_PROPERTY function is similar to the PROPERTY function except that it allows an optional `property-specific-argument` string parameter to be specified. The interpretation of `property-specific-argument` depends on the property ID or name specified in the first argument.

Privileges

No privileges are required to execute this function.

NULL is returned if you specify an invalid parameter value.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Execute the following statement to determine whether the xp_cmdshell system procedure can be used on the current connection without requiring a key:

```
SELECT EXTENDED_PROPERTY( 'HasSecuredFeature', 'cmdshell' );
```

If the CMDSHELL feature is not secured, the statement returns No. If the CMDSHELL feature is secured and a secured feature key is required to access the feature, the statement returns Yes.

Execute the following statement to determine whether there is a secured feature key that allows access to the CMDSHELL feature:

```
SELECT EXTENDED_PROPERTY( 'HasSecureFeatureKey', 'cmdshell' );
```

If there is no secured feature key available, the statement returns No. If there is a secured feature key that would permit access to this feature, the statement returns Yes.

Execute the following statement to determine whether the current connection can perform BACKUP and RESTORE statements without requiring a key..

```
SELECT EXTENDED_PROPERTY( 'HasSecuredFeature', 'backup,restore' );
```

If none of the features are secured, the statement returns No. If any feature is secured, the statement returns Yes. To determine if both features are secured, execute the following statement and check both results.

```
SELECT EXTENDED_PROPERTY( 'HasSecuredFeature', 'backup' ) AS [backup],
    EXTENDED_PROPERTY( 'HasSecuredFeature', 'restore' ) AS [restore];
```

Execute the following statement to determine whether there is a secure feature key that allows the current connection to perform BACKUP and RESTORE statements.

```
SELECT EXTENDED_PROPERTY( 'HasSecureFeatureKey', 'backup,restore' );
```

If there is a key that enables both features, the statement returns Yes. If any feature cannot be enabled by at least one key, the statement returns No.

Related Information

List of Database Server Properties
PROPERTY Function [System] [page 490]

1.3.2.88 EXTRACT Function [Date and Time]

Returns a date part from a DATE, TIME, TIMESTAMP, or TIMESTAMP WITH TIME ZONE expression.

Syntax

```
EXTRACT( date-part FROM timestamp-expression )
```

Parameters

date-part

The date part to be returned. The valid values are YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, and TIMEZONE_MINUTE.

timestamp-expression

The DATE, TIME, or TIMESTAMP or TIMESTAMP WITH TIME ZONE value.

Returns

If `date-part` is SECOND, then the function returns a string that includes the fractional second (up to microsecond precision). For all other `date-part` values, the function returns an INTEGER.

Remarks

The EXTRACT function is similar to the DATEPART function but not completely. The EXTRACT function accepts only a subset of date parts. Also, the two functions return different values when `date-part` is SECOND.

Date parts YY, MM, DD, HH, MI, SS, TZH, and TZM may also be used but do not conform to the SQL Standard.

Standards

ANSI/ISO SQL Standard

Core feature.

Example

The following statement returns 56.789000:

```
SELECT EXTRACT( SECOND FROM '2015-07-01 12:34:56.789000' );
```

The following statement returns 5:

```
SELECT EXTRACT( TIMEZONE_HOUR FROM CAST('2015-07-01 12:34:56.789000 +05:30' AS
TIMESTAMP WITH TIME ZONE) );
```

The following statement returns 30:

```
SELECT EXTRACT( TIMEZONE_MINUTE FROM CAST('2015-07-01 12:34:56.789000 +05:30' AS
TIMESTAMP WITH TIME ZONE) );
```

The following statement returns 2021:

```
SELECT EXTRACT( YEAR FROM '21-07-01 12:34:56.345678' );
```

It does so since 21-07-01 represents July 1, 2021.

Related Information

Specifying Date Parts [page 214]

first_day_of_week Option
SET Statement [T-SQL] [page 1375]
MICROSECOND Function [Date and Time] [page 453]
DATEPART Function [Date and Time] [page 322]
MILLISECOND Function [Date and Time] [page 454]

1.3.2.89 FIRST_VALUE Function [Aggregate]

Returns values from the first row of a window.

Syntax

```
FIRST_VALUE( [ ALL ] expression [ { RESPECT | IGNORE } NULLS ] )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

expression

The expression to evaluate. For example, a column name.

Returns

Data type of the values from the first row of a window.

Remarks

The FIRST_VALUE function allows you to select the first value (according to some ordering) in a table, without having to use a self-join. This is valuable when you want to use the first value as the baseline in calculations.

The FIRST_VALUE function takes the first record from the window. Then, the `expression` is computed against the first record and results are returned.

If IGNORE NULLS is specified, the first non-NULL value of `expression` is returned. If RESPECT NULLS is specified (the default), the first value is returned whether or not it is NULL.

The FIRST_VALUE function is different from most other aggregate functions in that it can only be used with a window specification.

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. More information is provided in the `window-spec` definition of the WINDOW clause.

Standards

ANSI/ISO SQL Standard

Not in the standard. The software supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". Also, the software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the FIRST_VALUE function, combined with an outer reference.

Example

The following example returns the relationship, as a percentage, between each employee's salary and that of the most recently hired employee in the same department:

```
SELECT DepartmentID, EmployeeID,
       100 * Salary / ( FIRST_VALUE( Salary ) OVER (
                          PARTITION BY DepartmentID  ORDER BY StartDate DESC ) )
           AS percentage
    FROM GROUPO.Employees;
```

DepartmentID	EmployeeID	percentage
500	1658	100
500	1615	110.4284624
500	1570	138.8427097
500	1013	109.5851905
500	921	167.4497049
500	868	113.2393688
500	750	137.7344095
500	703	222.8679276
500	191	119.6642975
400	1751	100
400	1740	99.705647
400	1684	130.969936
400	1643	83.9734797

DepartmentID	EmployeeID	percentage
400	1607	175.1828989
400	1576	197.0164609
...	...	...

Employee 1658 is the first row for department 500, indicating that they are the most recent hire in that department and their percentage is 100%. Percentages for the remaining department 500 employees are calculated relative to that of employee 1658. For example, employee 1570 earns approximately 139% of what employee 1658 earns.

If another employee in the same department makes the same salary as the most recent hire, they will have a percentage of 100 as well.

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
Window Aggregate Functions
WINDOW Clause [page 1489]
LAST_VALUE Function [Aggregate] [page 431]

1.3.2.90 FLOOR Function [Numeric]

Returns the largest integer not greater than the given number.

Syntax

```
FLOOR( numeric-expression )
```

Parameters

numeric-expression

The value to be truncated, typically a fixed numeric type with non-zero scale or an approximate numeric type (DOUBLE, REAL, or FLOAT).

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

Standards

ANSI/ISO SQL Standard

The FLOOR function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns a Floor value of 123:

```
SELECT FLOOR (123);
```

The following statement returns a Floor value of 123:

```
SELECT FLOOR (123.45);
```

The following statement returns a Floor value of -124:

```
SELECT FLOOR (-123.45);
```

Related Information

CEILING Function [Numeric] [page 278]

1.3.2.91 GET_BIT Function [Bit Array]

Returns the value (1 or 0) of a specified bit in a bit array.

Syntax

```
GET_BIT( bit-expression, position )
```

Parameters

bit-expression

The bit array containing the bit.

position

The position of the bit for which to return the status.

Returns

BIT

Remarks

The positions in the array are counted from the left side, starting at 1.

If `position` exceeds the length of the array, 0 (false) is returned.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 1:

```
SELECT GET_BIT( '00110011' , 4 );
```

The following statement returns the value 0:

```
SELECT GET_BIT( '00110011' , 5 );
```

Related Information

Bitwise Operators [page 33]
SET_BIT Function [Bit Array] [page 548]
SET_BITS Function [Aggregate] [page 549]
sa_get_bits System Procedure [page 1596]

1.3.2.92 GET_IDENTITY Function [Miscellaneous]

Allocates values to an AUTOINCREMENT column. This is an alternative to using AUTOINCREMENT to generate numbers.

Syntax

```
GET_IDENTITY( table_name [ , number_to_allocate ] )
```

Parameters

table_name

A string indicating the name of the table, including, optionally, the owner name.

number_to_allocate

The number of values to reserve. The default is 1.

Returns

UNSIGNED BIGINT

Remarks

Using AUTOINCREMENT or GLOBAL AUTOINCREMENT is still the most efficient way to generate IDs, but this function is provided as an alternative. The function assumes that the table has an AUTOINCREMENT column

defined. It returns the next available value that would be generated for the table's AUTOINCREMENT column, and reserves that value so that no other connection will use it by default.

The function returns an error if the table is not found, and returns NULL if the table has no AUTOINCREMENT column. If there is more than one AUTOINCREMENT column, it uses the first one it finds.

`number_to_allocate` is the number of values to reserve. If `number_to_allocate` is greater than 1, the function also reserves the remaining values. The next allocation uses the current number plus the value of `number_to_allocate`. This allows the application to execute the GET_IDENTITY function less frequently. If `number_to_allocate` is 0, the next available value is returned without reserving any values.

No COMMIT is required after executing the GET_IDENTITY function, and so it can be called using the same connection that is used to insert rows. If ID values are required for several tables, they can be obtained using a single SELECT that includes multiple calls to the GET_IDENTITY function, as in the example.

The GET_IDENTITY function is a non-deterministic function; successive calls to it may return different values. The optimizer does not cache the results of the GET_IDENTITY function.

Privileges

You must have INSERT privilege on `table-name`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the next available value for the Customers table AUTOINCREMENT column (ID). The number returned and the following nine values are reserved:

```
SELECT GET_IDENTITY( 'GROUPO.Customers', 10 );
```

Related Information

Function Caching
CREATE TABLE Statement [page 1002]
ALTER TABLE Statement [page 742]
NUMBER Function [Miscellaneous] [page 481]

1.3.2.93 GETDATE Function [Date and Time]

Returns the current year, month, day, hour, minute, second, and fraction of a second.

Syntax

```
GETDATE( )
```

Returns

TIMESTAMP

Remarks

The accuracy is limited by the accuracy of the system clock.

The information the GETDATE function returns is equivalent to the information returned by the NOW function and the CURRENT TIMESTAMP special value.

Note

If the database is using a simulated time zone, the simulated time zone is used to calculate the results of this function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the system date and time:

```
SELECT GETDATE( );
```

Related Information

CURRENT DATE Special Value [page 90]
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
Expressions in SQL Statements [page 34]
ISDATE Function [Data Type Conversion] [page 425]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.3.2.94 GRAPHICAL_PLAN Function [Miscellaneous]

Returns the plan optimization strategy of a SQL statement in XML format, as a string.

Syntax

```
GRAPHICAL_PLAN(
string-expression
[ , statistics-level
[ , cursor-type
[ , update-status ] ] ] )
```

Parameters

string-expression

The SQL statement, which is commonly a SELECT statement but which may also be an UPDATE or DELETE statement.

statistics-level

An integer. `Statistics-level` can be one of the following values:

Value	Description
0	Optimizer estimates only (default).
2	Detailed statistics including node statistics.
3	Detailed statistics.

cursor-type

A cursor type, expressed as a string. Possible values are: asensitive, insensitive, sensitive, or keyset-driven. If `cursor-type` is not specified, asensitive is used by default.

update-status

A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor:

Value	Description
READ-ONLY	The cursor is read-only.
READ-WRITE (default)	The cursor can be read or written to.
FOR UPDATE	The cursor can be read or written to. This is exactly the same as READ-WRITE.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following Interactive SQL example passes a SELECT statement as a string parameter and returns the plan for executing the query. It saves the plan in the file `plan.saplan` which can be opened and read using Interactive SQL.

```
SELECT GRAPHICAL_PLAN(  'SELECT * FROM GROUPO.Departments WHERE DepartmentID >
100' );
OUTPUT TO 'plan.saplan' FORMAT TEXT QUOTE '' HEXADECIMAL ASIS;
```

The following statement returns a string containing the graphical plan for a keyset-driven, updatable cursor over the query `SELECT * FROM Departments WHERE GROUPO.DepartmentID > 100`. It also causes the

server to annotate the plan with actual execution statistics, in addition to the estimated statistics that were used by the optimizer.

```
SELECT GRAPHICAL_PLAN(
   'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100',
   2,
   'keyset-driven', 'for update' );
```

Related Information

Advanced: Query Execution Plans
PLAN Function [Miscellaneous] [page 488]
EXPLANATION Function [Miscellaneous] [page 378]

1.3.2.95 GREATER Function [Miscellaneous]

Returns the greater of two parameter values.

Syntax

```
GREATER( expression-1 , expression-2 )
```

Parameters

expression-1

The first parameter value to be compared.

expression-2

The second parameter value to be compared.

Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

Remarks

If the parameters are equal, the first is returned.

Variables defined as type TABLE REF are not supported for this function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 10:

```
SELECT GREATER( 10, 5 ) FROM SYS.DUMMY;
```

Related Information

LESSER Function [Miscellaneous] [page 438]

1.3.2.96 GROUPING Function [Aggregate]

Identifies whether a column in a GROUP BY operation result set is NULL because it is part of a subtotal row, or NULL because of the underlying data.

Syntax

```
GROUPING( group-by-expression )
```

Parameters

group-by-expression

An expression appearing as a grouping column in the result set of a query that uses a GROUP BY clause. This function can be used to identify subtotal rows added to the result set by a ROLLUP or CUBE operation.

Returns

1

Indicates that `group-by-expression` is NULL because it is part of a subtotal row. The column is not a prefix column for that row.

0

Indicates that `group-by-expression` is a prefix column of a subtotal row.

Standards

ANSI/ISO SQL Standard

The GROUPING function is part of optional ANSI/ISO SQL Language Feature T431, "Extended grouping capabilities".

Related Information

Detection of NULLs Using the GROUPING Function
The ROLLUP Clause
The CUBE Clause
GROUP BY GROUPING SETS
Detection of NULLs Using the GROUPING Function
SELECT Statement [page 1362]

1.3.2.97 HASH Function [String]

Returns the specified value in hashed form.

Syntax

```
HASH( expression [ , algorithm ] )
```

Parameters

expression

The value to be hashed. This parameter is case sensitive, even in case-insensitive databases.

algorithm

The algorithm to use for the hash. Possible values include: CRC32, MD5, SHA1, SHA1_FIPS, SHA256, SHA256_FIPS, SHA384, SHA384_FIPS, SHA512, SHA512_FIPS. By default, the MD5 algorithm is used. FIPS-certified algorithms require a separate license.

Returns

Following are the return types, depending on the algorithm used:

- CRC32 returns a hexadecimal string. Use the HEXTOINT function to convert the hexadecimal string to a 32-bit integer.
- MD5 returns a VARCHAR(32)
- SHA1 returns a VARCHAR(40)
- SHA1_FIPS returns a VARCHAR(40)
- SHA256 returns a VARCHAR(64)
- SHA256_FIPS returns a VARCHAR(64)
- SHA384 returns a VARCHAR(128)
- SHA384_FIPS returns a VARCHAR(128)
- SHA512 returns a VARCHAR(128)
- SHA512_FIPS returns a VARCHAR(128)

Remarks

Using a hash converts the value to a byte sequence that is unique to each value passed to the function.

When hashing a VARBIT column, you may want to first cast the column to string so that variable length VARBIT values hash to different results.

If the database server was started with the -fips option, the algorithm used, or the behavior, may be different, as follows:

- SHA1_FIPS is used if SHA1 is specified
- SHA256_FIPS is used if SHA256 is specified.
- SHA384_FIPS is used if SHA384 is specified.
- SHA512_FIPS is used if SHA512 is specified.
- an error is returned if MD5 or no algorithm is specified.
- the CRC32 algorithm, which is not FIPS-certified, is allowed in FIPS mode because it is not considered a cryptographic algorithm.

> ⚠ Caution
>
> All the algorithms are one-way hashes. It is not possible to re-create the original string from the hash.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a table called user_info to store information about the users of an application, including their user ID and password. One row is also inserted into the table. The password is hashed using the HASH function and the SHA256 algorithm. Storing hashed passwords in this way can be useful if you do not want to store passwords in clear text, yet you have an external application that needs to compare passwords.

```
CREATE TABLE user_info (
  employee_id   INTEGER NOT NULL PRIMARY KEY,
  user_name CHAR(80),
  user_pwd CHAR(80) );
INSERT INTO user_info
  VALUES ( '1', 's_phillips', HASH( 'mypass', 'SHA256' ) );
```

Related Information

String Functions [page 222]
-fips Database Server Option
HEXTOINT Function [Data Type Conversion] [page 400]

1.3.2.98 HEXTOBIN Function [Data Type Conversion]

Returns the LONG BINARY equivalent of a hexadecimal string.

Syntax

```
HEXTOBIN( hexadecimal-string )
```

Parameters

hexadecimal-string

The string to be converted to a binary string.

Returns

The HEXTOBIN function returns a LONG BINARY string. If the number of characters in the input is odd, it is left-padded with a zero. The length of the result is the length of the input string divided by 2. If the input string contains any non-hexadecimal characters, an error is returned.

Remarks

The HEXTOBIN function accepts string keycodes or variables consisting only of digits and the uppercase or lowercase letters A-F.

The BINTOHEX, CAST, CONVERT, HEXTOBIN, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns a binary string containing `0x313233`:

```
SELECT HEXTOBIN('313233');
```

Related Information

Converting to and from Hexadecimal Values [page 12]
BINTOHEX Function [Data Type Conversion] [page 258]

1.3.2.99 HEXTOINT Function [Data Type Conversion]

Returns the decimal integer equivalent of a hexadecimal string.

Syntax

```
HEXTOINT( hexadecimal-string )
```

Parameters

hexadecimal-string

The string to be converted to an integer.

Returns

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values.

The HEXTOINT function returns as INT the platform-independent SQL INTEGER equivalent of the hexadecimal string. The hexadecimal value represents a negative integer if the 8th digit from the right is one of the digits 8-9 and the uppercase or lowercase letters A-F and the previous leading digits are all uppercase or lowercase letter F. The following is not a valid use of HEXTOINT since the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

```
SELECT HEXTOINT( '0x0080000001' );
```

Remarks

The HEXTOINT function accepts string keycodes or variables consisting only of digits and the uppercase or lowercase letters A-F, with or without a 0x prefix. The following are all valid uses of HEXTOINT:

```
SELECT HEXTOINT( '0xFFFFFFFF' );
SELECT HEXTOINT( '0x00000100' );
SELECT HEXTOINT( '100' );
SELECT HEXTOINT( '0xffffffff80000001' );
```

The HEXTOINT function removes the 0x prefix, if present. If the data exceeds 8 digits, it must represent a value that can be represented as a signed 32-bit integer value.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 420:

```
SELECT HEXTOINT( '1A4' );
```

Related Information

Converting to and from Hexadecimal Values [page 12]
INTTOHEX Function [Data Type Conversion] [page 424]

1.3.2.100 HOUR Function [Date and Time]

Returns the hour component of a TIMESTAMP value.

Syntax

```
HOUR( timestamp-expression )
```

Parameters

timestamp-expression

A TIMESTAMP value.

Returns

SMALLINT

Remarks

The value returned is the hour portion of the TIMESTAMP expression, a SMALLINT value between 0 and 23.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' );
```

1.3.2.101 HOURS Function [Date and Time]

Manipulates a TIMESTAMP or returns the number of hours between two TIMESTAMP values.

Syntax

Return number of hours between midnight 0000-02-29 and a TIMESTAMP value

```
HOURS ( timestamp-expression )
```

Return number of hours between two TIMESTAMP values

```
HOURS ( timestamp-expression , timestamp-expression )
```

Add hours to a TIMESTAMP

```
HOURS ( time-or-timestamp-expression , integer-expression )
```

Parameters

time-or-timestamp-expression

A value of type TIME or TIMESTAMP.

timestamp-expression

A value of type TIMESTAMP.

integer-expression

The number of hours to be added to `time-or-timestamp-expression`. If `integer-expression` is negative, the appropriate number of hours is subtracted from `time-or-timestamp-expression`.

Returns

INTEGER when returning the number of hours between two `time-or-timestamp-expression` values.

TIME or TIMESTAMP when adding time to a `time-or-timestamp-expression`

Remarks

The result of the HOURS function depends on its arguments.

Return number of hours since midnight 0000-02-29

If you pass a single `timestamp-expression` to the HOURS function, it will return the number of hours between midnight 0000-02-29 and `timestamp-expression` as an INTEGER.

> **i** Note
>
> 0000-02-29 is not meant to imply an actual date; it is the default TIMESTAMP value used by the HOURS function.

Return number of hours between two TIMESTAMP values

If you pass two TIMESTAMP values to the HOURS function, the function returns the integer number of hours between them.

Add hours to a TIMESTAMP

If you pass a TIMESTAMP value and an INTEGER value to the HOURS function, the function returns the TIMESTAMP result of adding the integer number of hours to `time-or-timestamp-expression` argument. Similarly, if you pass a TIME value as the first argument, a TIME value is returned as the result. This syntax does not support implicit conversion of the first argument. It may be necessary to explicitly cast the first argument to a DATE, TIME or TIMESTAMP value. If the first argument is a DATE, midnight is assumed for the time portion.

You can also use the DATEDIFF and DATEADD functions for these calculations.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements return the value 4, signifying that the second TIMESTAMP value is four hours after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT HOURS( '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
```

```
SELECT DATEDIFF( hour, '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
```

The following statement returns the value 17517342:

```
SELECT HOURS( '1998-07-13 06:07:12' );
```

The following statements return the datetime 1999-05-13 02:05:07.000. It is recommended that you use the second example (DATEADD).

```
SELECT HOURS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );
SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );
```

Related Information

DATEDIFF Function [Date and Time] [page 317]
DATEADD Function [Date and Time] [page 316]
CAST Function [Data Type Conversion] [page 275]

1.3.2.102 HTML_DECODE Function [Miscellaneous]

Decodes special character entities that appear in HTML literal strings.

Syntax

```
HTML_DECODE( string )
```

Parameters

string

Arbitrary literal string used in an HTML document.

Returns

LONG VARCHAR or LONG NVARCHAR.

Remarks

This function returns the string argument after making the appropriate substitutions. The following table contains a sampling of the acceptable character entities.

Characters	Substitution
&quot;	"
&#39;	'
&amp;	&
&lt;	<
&gt;	>
&#x*hexadecimal-number*;	Unicode codepoint, specified as a hexadecimal number. For example, ' returns a single apostrophe.
&#*decimal-number*;	Unicode codepoint, specified as a decimal number. For example, ™ returns the trademark symbol.

When a Unicode codepoint is specified, if the value can be converted to a character in the database character set, it is converted to a character. Otherwise, it is returned uninterpreted.

SQL Anywhere supports all character entity references specified in the HTML 4.01 Specification.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the string `<p>The piano was made by 'Steinway & Sons'.</p>`:

```
SELECT HTML_DECODE('&lt;p&gt;The piano was made ' ||
  'by &lsquo;Steinway &amp; Sons&rsquo;.&lt;/p&gt;');
```

The following statement returns the string `<p>It cost €85.000,000.</p>`:

```
SELECT HTML_DECODE('&lt;p&gt;It cost &euro;85.000,000.&lt;/p&gt;');
```

Related Information

Web Services Functions [page 221]

HTML_ENCODE Function [Miscellaneous] [page 407]
Web Services System Procedures [page 1512]

1.3.2.103 HTML_ENCODE Function [Miscellaneous]

Encodes special characters within strings to be inserted into HTML documents.

Syntax

```
HTML_ENCODE( string )
```

Parameters

string

Arbitrary string to be used in an HTML document.

Returns

LONG VARCHAR or LONG NVARCHAR.

Remarks

This function returns the string argument after making the following set of substitutions:

Characters	Substitution
"	&quot;
'	&#39;
&	&amp;
<	&lt;
>	&gt;
codes nn less than 0x20	&#xnn;

This function supports NCHAR inputs and/or outputs.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the string `'<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;> '`.

```
SELECT HTML_ENCODE('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">');
```

Related Information

Web Services Functions [page 221]
HTML_DECODE Function [Miscellaneous] [page 405]
Web Services System Procedures [page 1512]

1.3.2.104 HTTP_BODY Function [Web Service]

Returns the body of the HTTP request in binary form. For example, in a POST request, this is the raw POST data.

Syntax

```
HTTP_BODY( )
```

Parameters

None

Returns

LONG VARCHAR containing the body of the HTTP request in binary form; no character set conversion is performed on it.

Remarks

If the request body does not exist, or if the function is not called from a web service, a NULL value is returned.

This function is useful within the PHP external environment.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Web Services Functions [page 221]
sa_http_php_page System Procedure [page 1615]
sa_http_php_page_interpreted System Procedure [page 1616]
Web Services System Procedures [page 1512]

1.3.2.105 HTTP_DECODE Function [Web Service]

Decodes HTTP encoded strings. This is also known as URL decoding.

Syntax

```
HTTP_DECODE( string )
```

Parameters

string

Arbitrary string taken from a URL or URL encoded request body.

Returns

LONG VARCHAR or LONG NVARCHAR

Remarks

This function returns the string argument after replacing all character sequences of the form %nn, where nn is a hexadecimal value, with the character with code nn. In addition, all plus signs (+) are replaced with spaces.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the string http://test.sap.com:

```
SELECT HTTP_DECODE('http%3A%2F%2Ftest.sap.com')
```

Related Information

Web Services Functions [page 221]
HTTP_ENCODE Function [Web Service] [page 410]
Web Services System Procedures [page 1512]

1.3.2.106 HTTP_ENCODE Function [Web Service]

Encodes strings for use with HTTP. This is also known as URL encoding.

Syntax

```
HTTP_ENCODE( string )
```

Parameters

string

Arbitrary string to be encoded for HTTP transport.

Returns

LONG VARCHAR or LONG NVARCHAR

Remarks

This function returns the string argument after making the following set of substitutions. In addition, all characters with hexadecimal codes less than 20 or greater than 7E are replaced with %nn, where nn is the character code.

Character	Substitution
space	%20
"	%22
#	%23
%	%25
&	%26
,	%2C
+	%2B
;	%3B
<	%3C
>	%3E
[	%5B
\	%5C
]	%5D
^	%5E
`	%60
{	%7B
|	%7C
}	%7D
character codes nn that are less than 0x20 and greater than 0x7E	%nn

This function supports NCHAR inputs and/or outputs.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the string `/opt%26id=123%26text='oid:c%09d%20ef'`:

```
SELECT HTTP_ENCODE('/opt&id=123&text=''oid:c\x09d ef''')
```

Related Information

Web Services Functions [page 221]
HTTP_DECODE Function [Web Service] [page 409]
Web Services System Procedures [page 1512]

1.3.2.107 HTTP_HEADER Function [Web Service]

Returns the value of an HTTP request header.

Syntax

```
HTTP_HEADER( header-field-name [ , instance ] )
```

Parameters

header-field-name

The name of an HTTP request header field.

instance

The instance of the header to retrieve. If more than one header has the same name, then the instance is the number of the field instance. A value of 0 or NULL returns the most recent instance of the header. The default is 0.

Returns

LONG VARCHAR.

Remarks

This function returns the value of the named HTTP request header field, or NULL if it does not exist or if it is not called from an HTTP service. It is used when processing an HTTP request via a web service.

Some headers that may be of interest when processing an HTTP web service request include the following:

Cookie

The cookie value(s), if any, stored by the client that are associated with the requested URI.

Referer

The URL of the page (for example, `http://documents.sample.com:80/index.html`) that contained the link to the requested URI.

Host

The Internet host name or IP address and port number of the resource being requested, as obtained from the original URI given by the user or referring resource (for example, `webserver.sample.com:8082`).

User-Agent

The name of the client application (for example, `Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0`).

Accept-Encoding

A list of encodings for the response that are acceptable to the client application (for example, `gzip, deflate`).

More information about these headers is available at HTTP Header Field Definitions .

The following special headers allow access to the elements within the request line of a client request.

@HttpMethod

Returns the type of request being processed. Possible values include DELETE, HEAD, GET, PUT, or POST.

@HttpURI

The full URI of the request, as it was specified in the HTTP request (for example, `/myservice?&id=-123&version=109&lang=en`).

@HttpVersion

The HTTP version of the request (for example, `HTTP/1.0`, or `HTTP/1.1`).

@HttpQueryString

Returns the query portion of the requested URI if it exists (for example, `id=-123&version=109&lang=en`).

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement retrieves the fifth instance of the Cookie header value when used within a stored procedure that is called by an HTTP web service:

```
SET cookie_header = HTTP_HEADER( 'Cookie', 5 );
```

The following statement displays the name and values of the HTTP request headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:

```
BEGIN
  declare header_name long varchar;
  declare header_value long varchar;
  set header_name  = NULL;
header_loop:
  LOOP
    SET header_name = NEXT_HTTP_HEADER( header_name );
    IF header_name IS NULL THEN
      LEAVE header_loop
    END IF;
    SET header_value = HTTP_HEADER( header_name );
    MESSAGE 'HEADER: ', header_name, '=',
            header_value TO CONSOLE;
  END LOOP;
END;
```

Related Information

How to Access Client-supplied HTTP Variables and Headers
Web Services Functions [page 221]
NEXT_HTTP_HEADER Function [Web Service] [page 472]
sa_set_http_header System Procedure [page 1711]
sa_http_header_info System Procedure [page 1613]
Web Services System Procedures [page 1512]

1.3.2.108 HTTP_RESPONSE_HEADER Function [Web Service]

Returns the value of an HTTP response header.

Syntax

```
HTTP_RESPONSE_HEADER( header-field-name [ , instance ] )
```

Parameters

header-field-name

The name of an HTTP response header field.

instance

The instance of the header to retrieve. If more than one header has the same name, then the instance is the number of the field instance. A value of 0 or NULL returns the most recent instance of the header. The default is 0.

Returns

LONG VARCHAR

Remarks

This function returns the value of the named HTTP response header field, or NULL if a header for the given `header-field-name` does not exist or if it is not called from an HTTP service.

Some headers that may be of interest when processing an HTTP web service response include the following:

Connection

The Connection field allows the sender to specify options that are desired for that particular connection. In an SQL Anywhere HTTP server response, the option is always "close".

Content-Length

The Content-Length field indicates the size of the response body, in decimal number of octets.

Content-Type

The Content-Type field indicates the media type of the body sent to the recipient. For example: text/xml

Date

The Date field represents the date and time at which the response was originated.

Expires

The Expires field gives the date and time after which the response is considered stale.

Location

The Location field is used to redirect the recipient to a location for completion of the request or identification of a new resource.

Server

The Server field contains information about the software used by the origin server to handle the request. In an SQL Anywhere HTTP server response, the web server name together with the version number is returned.

Transfer-Encoding

The Transfer-Encoding field indicates what (if any) type of transformation has been applied to the message body to safely transfer it between the sender and the recipient.

User-Agent

The User-Agent field contains information about the user agent originating the request. In an SQL Anywhere HTTP server response, the web server name together with the version number is returned.

WWW-Authenticate

The WWW-Authenticate field is included in 401 (Unauthorized) response messages.

More information about these headers is available at HTTP Header Field Definitions .

The following special header allows access to the status within the response of a server response.

@HttpStatus

Returns the status code of the processed request.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement displays the name and values of the HTTP response headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:

```
BEGIN
  declare header_name long varchar;
  declare header_value long varchar;
  set header_name = NULL;
header_loop:
  LOOP
    SET header_name = NEXT_HTTP_RESPONSE_HEADER( header_name );
    IF header_name IS NULL THEN
      LEAVE header_loop
    END IF;
    SET header_value = HTTP_RESPONSE_HEADER( header_name );
    MESSAGE 'RESPONSE HEADER: ', header_name, '=', header_value TO CONSOLE;
  END LOOP;
```

Related Information

How to Access Client-supplied HTTP Variables and Headers
Web Services Functions [page 221]
NEXT_HTTP_RESPONSE_HEADER Function [Web Service] [page 474]
sa_set_http_header System Procedure [page 1711]

Web Services System Procedures [page 1512]

1.3.2.109 HTTP_VARIABLE Function [Web Service]

Returns the value of an HTTP variable.

Syntax

```
HTTP_VARIABLE( var-name [ , instance [ , attribute ] ] )
```

Parameters

var-name

The name of an HTTP variable.

instance

If more than one variable has the same name, the instance number of the field instance, or NULL to get the first one. Useful for SELECT lists that permit multiple selections.

attribute

In a multi-part request, the attribute can specify a header field name, which returns the value of the header for the multi-part section.

When an attribute is not specified, the returned value is %-decoded and character-set translated to the database character set. UTF %-encoded data is supported in this mode.

The attribute can also be one of the following modes:

'@BINARY'

Returns a x-www-form-urlencoded binary data value. This mode indicates that the returned value is %-decoded and not character-set translated. UTF-8 %-encoding is not supported in this mode since %-encoded data are simply decoded into their equivalent byte representation.

'@TRANSPORT'

Returns the raw HTTP transport form of the value, where %-encodings are preserved.

Returns

LONG VARCHAR.

Remarks

This function returns the value of the named HTTP variable. It is used when processing an HTTP request within a web service.

If `var-name` does not exist, the return value is NULL.

When the web service request is a POST, and the variable data is posted as multipart/form-data, the HTTP server receives HTTP headers for each individual variable. When the `attribute` parameter is specified, the HTTP_VARIABLE function returns the associated multipart/form-data header value from the POST request for the particular variable. For a variable representing a file, an attribute of Content-Disposition, Content-Type, and @BINARY would return the filename, media-type, and file contents respectively.

Normally, all input data goes through character set translation between the client (for example, a browser) character set, and the character set of the database. However, if @BINARY is specified for `attribute`, the variable value is returned without going through character set translation or %-decoding. This may be useful when receiving binary data, such as image data, from a client.

This function returns NULL when the specified instance does not exist or when the function is called from outside of an execution of a web service.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement retrieves the values of the HTTP variables indicated in the sample URL when used within a stored procedure that is called by an HTTP web service:

```
-- http://sample.com/demo/ShowDetail?product_id=300&customer_id=101
BEGIN
  DECLARE v_customer_id LONG VARCHAR;
  DECLARE v_product_id LONG VARCHAR;
  SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
  SET v_product_id = HTTP_VARIABLE( 'product_id' );
  CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;
```

The following statements request the Content-Disposition and Content-Type headers of the image variable when used within a stored procedure that is called by an HTTP web service:

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
```

The following statement requests the value of the image variable in its current character set without going through character set translation when used within a stored procedure that is called by an HTTP web service:

```
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

Related Information

How to Access Client-supplied HTTP Variables and Headers
Web Services Functions [page 221]
NEXT_HTTP_VARIABLE Function [Web Service] [page 475]
sa_http_variable_info System Procedure [page 1618]
Web Services System Procedures [page 1512]

1.3.2.110 IDENTITY Function [Miscellaneous]

Generates integer values, starting at 1, for each successive row in a query.

Syntax

```
IDENTITY( expression )
```

Parameters

expression

An expression. The expression is parsed, but is ignored during the execution of the function.

Returns

INT

Remarks

The capability of the IDENTITY function is the same as the capability of the NUMBER function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns a sequentially numbered list of employees:

```
SELECT IDENTITY( 10 ), Surname FROM GROUPO.Employees;
```

Related Information

NUMBER Function [Miscellaneous] [page 481]

1.3.2.111 IFNULL Function [Miscellaneous]

Evaluates whether one expression is NULL and returns a value.

Syntax

```
IFNULL( expression-1 , expression-2 [ , expression-3 ] )
```

Parameters

expression-1

The expression to be evaluated. Its value determines whether `expression-2` or `expression-3` is returned.

expression-2

The return value if `expression-1` is NULL.

expression-3

The return value if `expression-1` is not NULL.

Returns

The data type returned depends on the data type of `expression-2` and `expression-3`.

Remarks

If the first expression is the NULL value, then the value of the second expression is returned. If the first expression is not NULL, the value of the third expression is returned. If the first expression is not NULL and there is no third expression, NULL is returned.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value -66:

```
SELECT IFNULL( NULL, -66 );
```

The following statement returns `NULL`, because the first expression is not NULL and there is no third expression:

```
SELECT IFNULL( -66, -66 );
```

1.3.2.112 INDEX_ESTIMATE Function [Miscellaneous]

Returns selectivity estimates from the index as a percentage calculated by the query optimizer, based on specified parameters.

Syntax

```
INDEX_ESTIMATE( column-name , value [ , relation-string ] )
```

Parameters

column-name

The column used in the estimate.

value

The value to which the column is compared.

relation-string

The comparison operator used for the comparison, enclosed in single quotes. Possible values for this parameter are: '=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' , and '!>'. The default is '='.

Returns

REAL

Remarks

This function returns selectivity estimates from the index for the predicate `column-name relation-string value`. If `value` is NULL and the relation string is '=', the selectivity is for the predicate `column-name IS NULL`. If `value` is NULL and the relation string is '!=' or '<>', the selectivity is for the predicate `column-name IS NOT NULL`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the percentage of EmployeeID values estimated to be greater than 200:

```
SELECT INDEX_ESTIMATE( EmployeeID, 200, '>' )
FROM GROUPO.Employees;
```

Related Information

ESTIMATE Function [Miscellaneous] [page 366]
ESTIMATE_SOURCE Function [Miscellaneous] [page 367]
EXPERIENCE_ESTIMATE Function [Miscellaneous] [page 376]

1.3.2.113 INSERTSTR Function [String]

Inserts a string into another string at a specified position.

Syntax

```
INSERTSTR( integer-expression , string-expression-1 , string-expression-2 )
```

Parameters

integer-expression

The position after which the string is to be inserted. Use zero to insert a string at the beginning.

string-expression-1

The string into which the other string is to be inserted.

string-expression-2

The string to be inserted.

Returns

LONG BINARY, LONG VARCHAR, or LONG NVARCHAR, depending on the data type of the input expressions.

Remarks

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value `backoffice`:

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```

Related Information

String Functions [page 222]
STUFF Function [String] [page 577]

1.3.2.114 INTTOHEX Function [Data Type Conversion]

Returns a string containing the hexadecimal equivalent of an integer.

Syntax

```
INTTOHEX( integer-expression )
```

Parameters

integer-expression

The integer to be converted to hexadecimal.

Returns

VARCHAR

Remarks

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 0000009c:

```
SELECT INTTOHEX( 156 );
```

Related Information

Converting to and from Hexadecimal Values [page 12]
HEXTOINT Function [Data Type Conversion] [page 400]

1.3.2.115 ISDATE Function [Data Type Conversion]

Tests if a string argument can be converted to a date.

Syntax

```
ISDATE( string )
```

Parameters

string

The string to be analyzed to determine if the string represents a valid date.

Returns

INT

Remarks

If a conversion is possible, the function returns 1; otherwise, 0 is returned. If the argument is NULL, 0 is returned.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example imports data from an external file into the sample database, exports rows which contain invalid values, and copies the remaining rows to a permanent table:

```
CREATE GLOBAL TEMPORARY TABLE MyData(
      person VARCHAR(100),
      birth_date VARCHAR(30),
      height_in_cms VARCHAR(10)
   ) ON COMMIT PRESERVE ROWS;
   LOAD TABLE MyData FROM 'exported.dat';
   UNLOAD
      SELECT * FROM MyData
      WHERE ISDATE( birth_date ) = 0
  OR ISNUMERIC( height_in_cms ) = 0
   TO 'badrows.dat';
   INSERT INTO PermData
      SELECT person, birth_date, height_in_cms
      FROM MyData
      WHERE ISDATE( birth_date ) = 1
  AND ISNUMERIC( height_in_cms ) = 1;
   COMMIT;
   DROP TABLE MyData;
```

Related Information

CURRENT TIME Special Value [page 95]

CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
NOW Function [Date and Time] [page 478]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]
TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
UTC TIMESTAMP Special Value [page 113]
Time Zone Management
Creating Simulated Time Zones (SQL)

1.3.2.116 ISENCRYPTED Function [System]

Determines if a string is encrypted using the ENCRYPT function and the specified key.

Syntax

```
ISENCRYPTED( string , key [ , algorithm ] )
```

Parameters

string

The string to be analyzed to determine if it is encrypted. This parameter is case sensitive, even in case-insensitive databases.

key

The encryption key used to encrypt the `string`. This parameter is case sensitive, even in case-insensitive databases.

algorithm

This optional parameter specifies the algorithm used when the `string` was encrypted. Supported algorithms include: AES, AES256, AES_FIPS, and AES256_FIPS.

You can specify one of the FIPS-certified algorithms for `algorithm` on any platform that supports FIPS-certified encryption.

If `algorithm` is not specified, AES is used by default. If the database server was started using the -fips server option, the default is AES_FIPS.

Returns

INT

Remarks

ISENCRYPTED returns 1 when the input string is encrypted with the specified key; otherwise it returns 0.

> **i** Note
>
> For the ISENCRYPTED function to return meaningful results, data must be encrypted using the ENCRYPT function with AES/AES256 and must not use FORMAT=RAW.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 1 because the input string is encrypted:

```
SELECT ISENCRYPTED( ENCRYPT ('test_string', 'key' ), 'key');
```

Related Information

ENCRYPT Function [String] [page 350]
DECRYPT Function [String] [page 340]

1.3.2.117 ISNULL Function [Miscellaneous]

Returns the first non-NULL expression from a list. This function is identical to the COALESCE function.

Syntax

```
ISNULL( expression , expression [ , ... ] )
```

Parameters

expression

An expression to be tested against NULL.

At least two expressions must be passed into the function, and all expressions must be comparable.

Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the first non-NULL expression from the list. If the database server cannot find a common comparison type, then an error is returned.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value -66:

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 );
```

Related Information

COALESCE Function [Miscellaneous] [page 283]

1.3.2.118 ISNUMERIC Function [Miscellaneous]

Determines if the argument is a valid number.

Syntax

```
ISNUMERIC( value )
```

Parameters

value

The input value to be analyzed to determine if it represents a valid number.

Returns

INT

Remarks

ISNUMERIC returns 1 when `value` evaluates to a valid integer or double-precision floating-point number; otherwise, it returns 0.

If `value` has an integer, floating-point, or other numeric type, then ISNUMERIC returns 1. Examples are INTEGER, FLOAT(12), DOUBLE, and NUMERIC(15,3).

If `value` is a string literal or a hexadecimal literal, or has any non-numeric type, then `value` must conform to the following.

The accepted form of a number is: `[+|-]digit*[.digit*][[E|e][+|-]digitdigit*]`.

Specifically, the accepted form of a number is an optional + or - sign, followed by zero or more digits, followed by an optional decimal point and zero or more digits, followed by an optional exponent. The exponent is a letter E or e, followed by an optional + or - sign, followed by one or more digits.

At least one digit is required before the optional exponent. No other characters are allowed. Leading or trailing spaces are permitted.

Any other input will result in a return of 0.

The letters D or d cannot be used as an alternate for E or e.

If the absolute value of the number exceeds the largest double-precision floating-point value that can be represented (approximately, 1.7976931348623158e+308), then 0 is returned.

However, if the absolute value of the number is less than the smallest double-precision floating-point value that can be represented (approximately, 2.2250738585072014e-308), then 1 is returned and the value is treated as if 0 had been specified.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns 1 for both ISNUMERIC tests. The literals used in this example represent the integer vaue 43.

```
SELECT '  +43', ISNUMERIC('  +43'),
    CAST(0x20202b3433 AS VARCHAR(5)), ISNUMERIC(0x20202b3433);
```

The following example imports data from an external file, exports rows that contain invalid values, and copies the remaining rows to a permanent table. In this example, the ISNUMERIC statement validates that the values in height_in_cms values are numeric.

```
CREATE GLOBAL TEMPORARY TABLE MyData(
      person VARCHAR(100),
      birth_date VARCHAR(30),
      height_in_cms VARCHAR(10)
    ) ON COMMIT PRESERVE ROWS;
LOAD TABLE MyData
    FROM 'exported.dat';
UNLOAD
    SELECT *
        FROM MyData
        WHERE ISDATE( birth_date ) = 0 OR ISNUMERIC( height_in_cms ) = 0
    TO 'badrows.dat';
INSERT INTO PermData
    SELECT person, birth_date, height_in_cms
        FROM MyData
        WHERE ISDATE( birth_date ) = 1 AND ISNUMERIC( height_in_cms ) = 1;
COMMIT;
DROP TABLE MyData;
```

1.3.2.119 LAST_VALUE Function [Aggregate]

Returns values from the last row of a window.

Syntax

```
LAST_VALUE( [ ALL ] expression [ { RESPECT | IGNORE } NULLS ] )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

expression

The expression to evaluate. For example, a column name.

Returns

Data type of the argument.

Remarks

The LAST_VALUE function allows you to select the last value (according to some ordering) in a table, without having to use a self-join. This is valuable when you want to use the last value as the baseline in calculations.

The LAST_VALUE function takes the last record from the partition after doing the ORDER BY. Then, the `expression` is computed against the last record and results are returned.

If IGNORE NULLS is specified, the last non-NULL value of `expression` is returned. If RESPECT NULLS is specified (the default), the last value is returned whether or not it is NULL.

The LAST_VALUE function is different from most other aggregate functions in that it can only be used with a window specification.

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the `window-spec` definition for the WINDOW clause.

Standards

ANSI/ISO SQL Standard

Not in the standard. The software supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". Also, the software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the LAST_VALUE function, combined with an outer reference.

Example

The following example returns the salary of each employee, plus the name of the employee with the highest salary in the same department:

```
SELECT GivenName + ' ' + Surname AS employee_name,
       Salary, DepartmentID,
       LAST_VALUE( employee_name ) OVER Salary_Window AS highest_paid
FROM GROUPO.Employees
WINDOW Salary_Window AS ( PARTITION BY DepartmentID ORDER BY Salary
                          RANGE BETWEEN UNBOUNDED PRECEDING
                          AND UNBOUNDED FOLLOWING );
```

employee_name	Salary	DepartmentID	highest_paid
Michael Lynch	24903	500	Jose Martinez
Joseph Barker	27290	500	Jose Martinez
Sheila Romero	27500	500	Jose Martinez
Felicia Kuo	28200	500	Jose Martinez
Jeannette Bertrand	29800	500	Jose Martinez
Jane Braun	34300	500	Jose Martinez
Anthony Rebeiro	34576	500	Jose Martinez
Charles Crowley	41700	500	Jose Martinez
Jose Martinez	55500.8	500	Jose Martinez
Doug Charlton	28300	400	Scott Evans
Elizabeth Lambert	29384	400	Scott Evans
Joyce Butterfield	34011	400	Scott Evans
Robert Nielsen	34889	400	Scott Evans
Alex Ahmed	34992	400	Scott Evans
Ruth Wetherby	35745	400	Scott Evans
...	...	...	...

Jose Martinez makes the highest salary in department 500, and Scott Evans makes the highest salary in department 400.

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
Window Aggregate Functions
WINDOW Clause [page 1489]
FIRST_VALUE Function [Aggregate] [page 385]

1.3.2.120 LCASE Function [String]

Converts all characters in a string to lowercase.

Syntax

```
LCASE( string-expression )
```

Parameters

string-expression

The string to be converted to lowercase.

Returns

LONG NVARCHAR when used on NCHAR data and LONG VARCHAR when used on CHAR data if the database collation is UCA.Otherwise, the data type is the same as the input data type.

UltraLite: The returned data type is the same as the input data type.

Remarks

The LCASE function is identical to the LOWER function.

Standards

ANSI/ISO SQL Standard

Not in the standard. The equivalent function LOWER is a Core Feature.

Example

The following statement returns the value `chocolate`:

```
SELECT LCASE( 'ChoCOlatE' );
```

Related Information

String Functions [page 222]
LOWER Function [String] [page 446]
UCASE Function [String] [page 607]
UPPER Function [String] [page 611]

1.3.2.121 LEFT Function [String]

Returns multiple characters from the beginning of a string.

Syntax

```
LEFT( string-expression , integer-expression )
```

Parameters

string-expression

The string.

integer-expression

The number of characters to return.

Returns

LONG VARCHAR or LONG NVARCHAR

UltraLite: LONG VARCHAR

Remarks

If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned may be greater than the specified number of characters.

You can specify an `integer-expression` that is larger than the value in the argument string expression. In this case, the entire value is returned.

Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the first 5 characters of each Surname value in the Customers table:

```
SELECT LEFT( Surname, 5) FROM GROUPO.Customers;
```

Related Information

String Functions [page 222]
RIGHT Function [String] [page 532]

1.3.2.122 LENGTH Function [String]

Returns the number of characters in the specified string.

Syntax

```
{ LENGTH | LEN } ( string-expression )
```

Parameters

string-expression

The string.

Returns

INT

Remarks

Use this function to determine the length of a string. For example, specify a column name for `string-expression` to determine the length of values in the column.

If the string contains multibyte characters, and the proper collation is being used, LENGTH returns the number of characters, not the number of bytes. If the string is of data type BINARY, the LENGTH function behaves as the BYTE_LENGTH function.

You can use the LENGTH function and the CHAR_LENGTH function interchangeably for CHAR, VARCHAR, LONG VARCHAR, and NCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types. This function supports NCHAR inputs and/or outputs.

UltraLite does not support the function name short form *LEN*.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

The LENGTH function is not in the standard; however, its semantics are identical to those of the CHAR_LENGTH function in the ANSI/ISO SQL Standard. Using LENGTH over a string expression of type NCHAR comprises part of optional ANSI/ISO SQL Language Feature F421.

Example

The following statement returns the value 9:

```
SELECT LENGTH( 'chocolate' );
```

Related Information

International Languages and Character Sets
String Functions [page 222]
BYTE_LENGTH Function [String] [page 267]

1.3.2.123 LESSER Function [Miscellaneous]

Returns the lesser of two parameter values.

Syntax

```
LESSER( expression-1 , expression-2 )
```

Parameters

expression-1

The first parameter value to be compared.

expression-2

The second parameter value to be compared.

Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

Remarks

If the parameters are equal, the first value is returned.

Variables defined as type TABLE REF are not supported for this function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 5:

```
SELECT LESSER( 10, 5 ) FROM SYS.DUMMY;
```

Related Information

GREATER Function [Miscellaneous] [page 395]

1.3.2.124 LIST Function [Aggregate]

Returns a delimited list of values for every row in a group.

Syntax

```
LIST( [ ALL | DISTINCT ] string-expression [, delimiter-string ] [ ORDER BY
order-by-expression [ ASC | DESC ] , ... ] )
```

UltraLite:

```
LIST( [ DISTINCT ] string-expression [ , delimiter-string ] )
```

Parameters

string-expression

A string expression, usually a column name. When ALL is specified (the default), for each row in the group, the value of `string-expression` is added to the result string, with values separated by `delimiter-string`. When DISTINCT is specified, only unique `string-expression` values are added.

UltraLite: For each row in the group, the value of `string-expression` is added to the result string, with values separated by `delimiter-string`.

delimiter-string

A delimiter string for the list items. The default setting is a comma. There is no delimiter if a value of NULL or an empty string is supplied. The `delimiter-string` must be a constant.

order-by-expression

Order the items returned by the function. There is no comma preceding this argument, which makes it easy to use in the case where no `delimiter-string` is supplied.

`order-by-expression` cannot be an integer literal. However, it can be a variable that contains an integer literal.

When an ORDER BY clause contains constants, they are interpreted by the optimizer and then replaced by an equivalent ORDER BY clause. For example, the optimizer interprets ORDER BY 'a' as ORDER BY expression.

A query block containing more than one aggregate function with valid ORDER BY clauses can be executed if the ORDER BY clauses can be logically combined into a single ORDER BY clause. For example, the following clauses:

```
ORDER BY expression1, 'a', expression2
```

```
ORDER BY expression1, 'b', expression2, 'c', expression3
```

are subsumed by the clause:

```
ORDER BY expression1, expression2, expression3
```

Returns

LONG VARCHAR
LONG NVARCHAR

> **i Note**
>
> UltraLite does not return LONG NVARCHAR.

Remarks

The LIST function returns the concatenation (with delimiters) of all the non-NULL values of X for each row in the group. If there does not exist at least one row in the group with a definite X-value, then LIST(X) returns the empty string.

NULL values and empty strings are ignored by the LIST function.

A LIST function cannot be used as a window function, but it can be used as input to a window function.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

The software supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the LIST function, combined with an outer reference.

Example

The following statement returns a list of all the street addresses for employees whose given name is Thomas:

```
SELECT LIST( Street ) FROM GROUPO.Employees
WHERE GivenName = 'Thomas';
```

The following statement returns lists of the names of cities delimited by semicolons and their state, organized by state :

```
SELECT LIST( DISTINCT City, ';' ), State FROM GROUPO.Employees
GROUP BY State;
```

The following statement lists employee IDs. Each row in the result set contains a comma-delimited list of employee IDs for a single department.

```
SELECT LIST( EmployeeID )
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

LIST(EmployeeID)
102,105,160,243,247,249,266,278,...
129,195,299,467,641,667,690,856,...
148,390,586,757,879,1293,1336,...
184,207,318,409,591,888,992,1062,...
191,703,750,868,921,1013,1570,...

The following statement sorts the employee IDs by the last name of the employee:

```
SELECT LIST( EmployeeID ORDER BY Surname ) AS "Sorted IDs"
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

Sorted IDs
1013,191,750,921,868,1658,...
1751,591,1062,1191,992,888,318,...
1336,879,586,390,757,148,1483,...
1039,129,1142,195,667,1162,902,...

The following statement returns semicolon-separated lists. Note the position of the ORDER BY clause and the list separator:

```
SELECT LIST( EmployeeID, ';' ORDER BY Surname ) AS "Sorted IDs"
```

```
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

Sorted IDs
1013;191;750;921;868;1658;703;...
1751;591;1062;1191;992;888;318;...
1336;879;586;390;757;148;1483;...
1039;129;1142;195;667;1162;902; ...
160;105;1250;247;266;249;445;...

Be sure to distinguish the previous statement from the following statement, which returns comma-separated lists of employee IDs sorted by a compound sort-key of `( Surname, ';' )`:

```
SELECT LIST( EmployeeID ORDER BY Surname, ';' ) AS "Sorted IDs"
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

UltraLite: The following statement returns all street addresses from the Employees table:

```
SELECT LIST( Street ) FROM GROUPO.Employees;
```

Related Information

sa_split_list System Procedure [page 1722]
ARRAY_AGG Function [Aggregate] [page 247]
Troubleshooting Database Upgrades: Aggregate Functions and Outer References

1.3.2.125 LOCATE Function [String]

Returns the position of one string within another.

Syntax

```
LOCATE( string-expression-1, string-expression-2 [, integer-expression ] )
```

Parameters

string-expression-1

The string to be searched.

string-expression-2

The string to be searched for.

This string is limited to 254 bytes.

integer-expression

The character position in the string to begin the search. The first character is position 1. If the starting offset is negative, the locate function returns the last matching string offset rather than the first. A negative offset indicates how much of the end of the string is to be excluded from the search. The number of bytes excluded is calculated as (-1 * offset) -1.

Returns

INT

Remarks

If `integer-expression` is specified, the search starts at that offset into the string.

The first string can be a long string (longer than 255 bytes), but the second is limited to 254 bytes. If a long string is given as the second argument, the function returns a NULL value. If the string is not found, 0 is returned. Searching for a zero-length string will return 1. If any of the arguments are NULL, the result is NULL.

If multibyte characters are used, with the appropriate collation, then the starting position and the return value may be different from the *byte* positions.

If arguments `string-expression-1` and `string-expression-2` are of binary data type, the LOCATE function behaves the same as the BYTE_LOCATE function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 8:

```
SELECT LOCATE(
   'office party this week - rsvp as soon as possible',
   'party',
   2 );
```

The following statement:

```
BEGIN
   DECLARE STR LONG VARCHAR;
   DECLARE POS INT;
   SET str = 'c:\test\functions\locate.sql';
   SET pos = LOCATE( str, '\', -1 );
   select str, pos,
      SUBSTR( str, 1, pos -1 ) AS path,
      SUBSTR( str, pos +1 ) AS filename;
END;
```

returns the following output:

str	pos	path	filename
c:\test\functions\locate.sql	18	c:\test\functions	locate.sql

Related Information

String Functions [page 222]
CHARINDEX Function [String] [page 282]

1.3.2.126 LOG Function [Numeric]

Returns the natural logarithm of a number.

Syntax

```
LOG( numeric-expression )
```

Parameters

numeric-expression

The number.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Remarks

The argument is an expression that returns the value of any built-in numeric data type.

Standards

ANSI/ISO SQL Standard

The ANSI/ISO SQL Standard defines the natural logarithm function using the keyword LN. The natural logarithm function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the natural logarithm of 50:

```
SELECT LOG( 50 );
```

Related Information

LOG10 Function [Numeric] [page 445]

1.3.2.127 LOG10 Function [Numeric]

Returns the base 10 logarithm of a number.

Syntax

```
LOG10( numeric-expression )
```

Parameters

numeric-expression

The number.

Returns

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the parameter is NULL, the result is NULL.

Remarks

The argument is an expression that returns the value of any built-in numeric data type.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the base 10 logarithm for 50:

```
SELECT LOG10( 50 );
```

Related Information

LOG Function [Numeric] [page 444]

1.3.2.128 LOWER Function [String]

Converts all characters in a string to lowercase.

Syntax

```
LOWER( string-expression )
```

Parameters

string-expression

The string to be converted to lowercase.

Returns

LONG NVARCHAR when used on NCHAR data
LONG VARCHAR when used on CHAR data if the database collation is UCA
Otherwise, the data type is the same as the input data type
UltraLite returns the same data type as the input data type

Remarks

The LCASE function is identical to the LOWER function.

Standards

ANSI/ISO SQL Standard

Core Feature. Using LOWER over an expression of type NCHAR comprises part of the optional Language Feature F421.

Example

The following statement returns the value `chocolate`:

```
SELECT LOWER( 'chOCOLate' );
```

Related Information

String Functions [page 222]
LCASE Function [String] [page 434]
UCASE Function [String] [page 607]
UPPER Function [String] [page 611]

1.3.2.129 LTRIM Function [String]

Removes leading blanks or specified characters from the string.

Syntax

```
LTRIM( string-expression [ , trim-char-set ] )
```

Parameters

string-expression

The string to be trimmed.

trim-char-set

The set of characters to trim.

Returns

VARCHAR
NVARCHAR
LONG VARCHAR
LONG NVARCHAR
UltraLite returns only VARCHAR or LONG VARCHAR

Remarks

By default, `trim-char-set` is the space character. You can specify the set of characters to be trimmed.

The actual length of the result is the length of the expression minus the number of characters removed. If all the characters are removed, the result is an empty string.

If the parameter can be null, the result can be null.

If the parameter is null, the result is the null value.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs and `trim-char-set`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

The TRIM specifications defined by the ANSI/ISO SQL Standard (LEADING and TRAILING) are supplied by the SQL Anywhere LTRIM and RTRIM functions respectively.

Example

The following statement returns the value `Test Message` with all leading blanks removed:

```
SELECT LTRIM( '     Test Message' );
```

The following statement returns the value def after the specified leading characters are removed:

```
SELECT LTRIM('abcabccbadef', 'abc' );
```

Related Information

String Functions [page 222]
RTRIM Function [String] [page 540]
TRIM Function [String] [page 601]

1.3.2.130 MAX Function [Aggregate]

Returns the maximum expression value found in each group of rows.

Syntax

Expression:

```
MAX( [ ALL | DISTINCT ] expression )
```

Window function:

```
MAX( [ ALL ] expression ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

UltraLite expression

```
MAX( [ DISTINCT ] expression )
```

Parameters

expression

The expression for which the maximum value is to be calculated. This is commonly a column name.

DISTINCT expression

Returns the same as MAX(`expression`), and is included for completeness.

Returns

The same data type as the argument.

Remarks

Rows where `expression` is NULL are ignored. Returns NULL for a group containing no rows.

Variables defined as type TABLE REF are not supported for this function.

For simple comparisons of two expressions, you can use the GREATER function.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. This function supports NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Core feature. When used as a window function, MAX comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions", nor does it permit the arguments of an aggregate function to include both a column reference from the query block containing the MAX function, combined with an outer reference.

Example

The following statement returns the value 138948.000, representing the maximum salary in the Employees table:

```
SELECT MAX( Salary )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
GREATER Function [Miscellaneous] [page 395]
MIN Function [Aggregate] [page 456]
Troubleshooting Database Upgrades: Aggregate Functions and Outer References

1.3.2.131 MEDIAN Function [Aggregate]

Computes the median of a numeric expression for a set of rows.

Syntax

Expression

```
MEDIAN( [ ALL | DISTINCT ] numeric-expression )
```

Window function

```
MEDIAN( [ ALL ]  numeric-expression) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

numeric-expression

The expression whose median is calculated over a set of rows.

DISTINCT clause

Eliminates duplicate values before computing the median of the unique values in the input.

ALL clause

Computes the median of all values (including duplicates) in the input. This is the default behavior.

Returns

The data type of the returned value is the same as that of the input value.

NULLs are ignored in the calculation of the median value. However, a NULL value is returned for a group that contains no rows.

Remarks

`numeric-expression` values can be of any numeric data type other than BIT.

The median of a finite list of numbers can be found by arranging all the observations from lowest value to highest value and picking the middle one. If there is an even number of observations, the median is not unique so MEDIAN returns the mean of the two middle values. At most, half the population have values less than the median, and half have values greater than the median. If both groups contain less than half the population, then some of the population is exactly equal to the median. For example, if `a < b < c`, then the median of the list `{a, b, c}` is b. If `a < b < c < d`, then the median of the list `{a, b, c, d}` is the mean of b and c (`(b + c)/2`).

If the result of the mean of the two middle elements has digits after the decimal place, they are truncated if the input data type cannot represent them. To avoid this truncation, cast the input to a numeric type that allows digits after the decimal place.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

`window-spec` can only be over a partition (it cannot contain a ROW or RANGE specification). DISTINCT is not supported if a WINDOW clause is used. CUBE, ROLLUP, and GROUPING SETS are supported with syntax 1.

Standards

ANSI/ISO SQL Standard

Not in the standard. However, window functions comprise optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The software supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software also does not permit the arguments of an aggregate function to include both a column reference from the query block containing the MEDIAN function, combined with an outer reference.

Example

The following statement returns the median salary from the Employees table:

```
SELECT MEDIAN( Salary ) FROM GROUPO.Employees;
```

The following statement returns the median salary by state from the Employees table:

```
SELECT EmployeeID, Surname, Salary, State,
  MEDIAN( Salary ) OVER Salary_Window
FROM GROUPO.Employees
WINDOW Salary_Window AS ( PARTITION BY State )
ORDER BY State, Surname;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
Numeric Data Types [page 143]
WINDOW Clause [page 1489]
SUM Function [Aggregate] [page 581]
COUNT Function [Aggregate] [page 300]

1.3.2.132 MICROSECOND Function [Date and Time]

Returns the microsecond component of a TIMESTAMP expression.

Syntax

```
MICROSECOND( timestamp-expression )
```

Parameters

timestamp-expression

A TIMESTAMP value.

Returns

INTEGER

Remarks

This function returns a value between 0 and 999999 that represents the fraction of a second (also referred to as a microsecond). This function is equivalent to `DATEPART( MICROSECOND, timestamp-expression )`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the microsecond value 789012:

```
SELECT MICROSECOND( '12:34:56.789012' );
```

Related Information

Specifying Date Parts [page 214]
first_day_of_week Option
SET Statement [T-SQL] [page 1375]
EXTRACT Function [Date and Time] [page 383]
DATEPART Function [Date and Time] [page 322]
MILLISECOND Function [Date and Time] [page 454]

1.3.2.133 MILLISECOND Function [Date and Time]

Returns the millisecond component of a TIMESTAMP expression.

Syntax

```
MILLISECOND( timestamp-expression )
```

Parameters

timestamp-expression

A TIMESTAMP value.

Returns

INTEGER

Remarks

This function returns a value between 0 and 999999 that represents the fraction of a second in milliseconds. If the timestamp contains fractions of a millisecond, then they are rounded down to the millisecond.

This function is equivalent to `DATEPART( MILLISECOND, timestamp-expression )`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the millisecond value 789:

```
SELECT MILLISECOND( '12:34:56.78901' );
```

Related Information

Specifying Date Parts [page 214]
first_day_of_week Option
SET Statement [T-SQL] [page 1375]
MICROSECOND Function [Date and Time] [page 453]
DATEPART Function [Date and Time] [page 322]
EXTRACT Function [Date and Time] [page 383]

1.3.2.134 MIN Function [Aggregate]

Returns the minimum expression value found in each group of rows.

Syntax

Expression

```
MIN( [ DISTINCT ] expression )
```

Window function

```
MIN( [ ALL ] expression ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

UltraLite expression

```
MIN( [ ALL | DISTINCT ] expression )
```

Parameters

expression

The expression for which the minimum value is to be calculated. This is commonly a column name.

DISTINCT expression

Returns the same as MIN(`expression`), and is included for completeness.

Returns

The same data type as the argument.

Remarks

Rows where `expression` is NULL are ignored. Returns NULL for a group containing no rows.

Variables defined as type TABLE REF are not supported for this function.

This function supports NCHAR inputs and/or outputs.

UltraLite: UltraLite does not support NCHAR inputs and/or outputs.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

Core feature. When used as a window function, MIN comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions", nor does it not permit the arguments of an aggregate function to include both a column reference from the query block containing the MIN function, combined with an outer reference.

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```
SELECT MIN( Salary )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
LESSER Function [Miscellaneous] [page 438]
WINDOW Clause [page 1489]
MAX Function [Aggregate] [page 449]
Troubleshooting Database Upgrades: Aggregate Functions and Outer References

1.3.2.135 MINUTE Function [Date and Time]

Returns the minute component of a TIMESTAMP value.

Syntax

```
MINUTE( timestamp-expression )
```

Parameters

timestamp-expression

The TIMESTAMP value.

Returns

SMALLINT

Remarks

The value returned is the minute portion of the TIMESTAMP expression, a SMALLINT value between 0 and 59.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 22:

```
SELECT MINUTE( '1998-07-13 12:22:34' );
```

1.3.2.136 MINUTES Function [Date and Time]

Manipulates a TIMESTAMP or returns the number of minute boundaries between two TIMESTAMP values.

Syntax

Return the number of minutes between midnight 0000-02-29 and a TIMESTAMP value

```
MINUTES( timestamp-expression )
```

Return the number of minutes between two TIMESTAMP values

```
MINUTES( timestamp-expression, timestamp-expression )
```

Add minutes to a TIMESTAMP value

```
MINUTES( timestamp-or-time-expression, integer-expression )
```

Parameters

timestamp-expression

An expression of type TIMESTAMP.

timestamp-or-time-expression

An expression of type TIME or TIMESTAMP.

integer-expression

The number of minutes to be added to `timestamp-or-time-expression`. If `integer-expression` is negative, the appropriate number of minutes is subtracted from `timestamp-or-time-expression`.

Returns

INTEGER, TIME, or TIMESTAMP, depending on the usage.

Remarks

The result of the MINUTES function depends on its arguments.

Return the number of minutes since midnight 0000-02-29

If you pass a single `timestamp-expression` to the MINUTES function, it will return the number of minute boundaries between midnight 0000-02-29 and `timestamp-expression` as an INTEGER.

> **i** Note
>
> 0000-02-29 is not meant to imply an actual date; it is the default date used by the MINUTES function.

Return the number of minutes between two TIMESTAMP values

If you pass two TIMESTAMP values to the MINUTES function, the function returns the integer number of minute boundaries between them.

Add minutes to a TIMESTAMP value

If you pass a TIMESTAMP value and an INTEGER value to the MINUTES function, the function returns the TIMESTAMP result of adding the integer number of minutes to `timestamp-expression` argument. Similarly, if the first argument to MINUTES is a TIME value, then the result is also a TIME value. This syntax does not support implicit conversion of the first argument. It may be necessary to explicitly cast the first argument to a DATE, TIME or TIMESTAMP value. If the first argument is of type DATE, midnight is assumed for the time portion.

Since MINUTES returns an integer, overflow can occur when used with TIMESTAMP values greater than or equal to 4083-03-23 02:08:00.

You can also use the DATEDIFF and DATEADD function for some of the calculations

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns identical values 240, signifying that the second TIMESTAMP value is 240 minutes after the first. It is recommended that you use DATEDIFF.

```
SELECT
  MINUTES( '1999-07-13 06:07:12',
           '1999-07-13 10:07:12' ),
  DATEDIFF( minute,
           '1999-07-13 06:07:12',
           '1999-07-13 10:07:12' );
```

The following statement returns the value 1051040527:

```
SELECT MINUTES( '1998-07-13 06:07:12' );
```

The following statements return the TIMESTAMP value 1999-05-12 21:10:07.000. The first statement requires an explicit cast of the literal string parameter. It is recommended that you use the second example (DATEADD).

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5);
SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```

Related Information

DATEDIFF Function [Date and Time] [page 317]
DATEADD Function [Date and Time] [page 316]
CAST Function [Data Type Conversion] [page 275]

1.3.2.137 MOD Function [Numeric]

Returns the remainder when one whole number is divided by another.

Syntax

```
MOD( dividend , divisor )
```

Parameters

dividend

The dividend, or numerator of the division.

divisor

The divisor, or denominator of the division.

Returns

- SMALLINT
- INT
- NUMERIC

Remarks

Division involving a negative dividend gives a negative or zero result. The sign of the divisor has no effect.

Standards

ANSI/ISO SQL Standard

The MOD function is part of optional ANSI/ISO SQL Language Feature T441.

Example

The following statement returns the value 2:

```
SELECT MOD( 5, 3 );
```

Related Information

REMAINDER Function [Numeric] [page 523]

1.3.2.138 MONTH Function [Date and Time]

Returns the month of the given date.

Syntax

```
MONTH( date-expression )
```

Parameters

date-expression

A value of type DATE.

Returns

SMALLINT

Remarks

The value returned is a number between 1 and 12, corresponding to the month of the given date.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 7:

```
SELECT MONTH( '1998-07-13' );
```

1.3.2.139 MONTHNAME Function [Date and Time]

Returns the name of the month from a date.

Syntax

```
MONTHNAME( date-expression )
```

Parameters

timestamp-expression

A TIMESTAMP value.

Returns

VARCHAR

Remarks

The MONTHNAME function returns a string, even if the result is numeric, such as 2 for the month of February.

In SQL Anywhere, English names are returned for an English locale, other names are returned when the locale is not English. For example, the Language (LANG) connection parameter can be used to specify a different language.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In an English locate, the following statement returns the value `September`:

```
SELECT MONTHNAME( '1998-09-05' );
```

In SQL Anywhere in a French locale, the value returned is `septembre`. In a Spanish locale, the value returned is `Septiembre`. Several locales are supported.

Related Information

DATENAME Function [Date and Time] [page 321]
DATEPART Function [Date and Time] [page 322]
Language (LANG) Connection Parameter

1.3.2.140 MONTHS Function [Date and Time]

Manipulates a TIMESTAMP or returns the number of month boundaries between two TIMESTAMP values.

Syntax

Return the number of months between 0000-02 and a TIMESTAMP value

```
MONTHS( timestamp-expression )
```

Return the number of months between two TIMESTAMP values

```
MONTHS( timestamp-expression, timestamp-expression )
```

Add months to a TIMESTAMP value

```
MONTHS( timestamp-expression, integer-expression )
```

Parameters

timestamp-expression

A date and time of type TIMESTAMP.

integer-expression

The integer number of months (of type SMALLINT) to be added to the `timestamp-expression`. If `integer-expression` is negative, the appropriate number of months is subtracted from `timestamp-expression`. If you supply an `integer-expression`, the `timestamp-expression` must be explicitly cast as a TIME, DATE or TIMESTAMP data type. If `timestamp-expression` is a TIME value, the current month is assumed.

Returns

INTEGER or TIMESTAMP, depending on the usage.

Remarks

The result of the MONTHS function depends on its arguments. The MONTHS function ignores hours, minutes, and seconds in its arguments.

Return the number of months since 0000-02

If you pass a single `timestamp-expression` to the MONTHS function, it will return the number of month boundaries between 0000-02 and `timestamp-expression` as an INTEGER.

> **i** Note
>
> 0000-02 is not meant to imply an actual date; it is the default date used by the MONTHS function.

Return the number of months between two TIMESTAMP values

If you pass two TIMESTAMP values to the MONTHS function, the function returns the integer number of month boundaries between them.

Add months to a TIMESTAMP value

If you pass a TIMESTAMP value and a SMALLINT value to the MONTHS function, the function returns the TIMESTAMP result of adding the integer number of months to `timestamp-expression`.

You can also use the DATEDIFF and DATEADD functions to perform some of these calculations.

The value of MONTHS is calculated from the number of first days of the month between the two dates.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements return the value 2, signifying that the second date is two months after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT MONTHS( '1999-07-13 06:07:12', '1999-09-13 10:07:12' );
```

```
SELECT DATEDIFF( month,
   '1999-07-13 06:07:12',
   '1999-09-13 10:07:12' );
```

The following statement returns the value 23981:

```
SELECT MONTHS( '1998-07-13 06:07:12' );
```

The following statements return the TIMESTAMP value 1999-10-12 21:05:07.000. It is recommended that you use the second example (DATEADD).

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5);
```

```
SELECT DATEADD( month, 5, '1999-05-12 21:05:07' );
```

Related Information

DATEDIFF Function [Date and Time] [page 317]
DATEADD Function [Date and Time] [page 316]
CAST Function [Data Type Conversion] [page 275]

1.3.2.141 NCHAR Function [String]

Returns an NCHAR string containing one character whose Unicode code point is given in the parameter, or NULL if the value is not a valid code point value.

Syntax

```
NCHAR( integer )
```

Parameters

integer

The number to be converted to the corresponding Unicode code point.

Returns

NVARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the ALEF Arabic letter, which is Unicode code point U+627:

```
SELECT NCHAR( 1575 );
```

Related Information

CONNECTION_EXTENDED_PROPERTY Function [String] [page 290]
TO_NCHAR Function [String] [page 591]
TO_CHAR Function [String] [page 589]
UNICODE Function [String] [page 608]
UNISTR Function [String] [page 609]

1.3.2.142 NEWID Function [Miscellaneous]

Generates a UUID (Universally Unique Identifier) value.

Syntax

NEWID( )

Parameters

There are no parameters associated with the NEWID function.

Returns

UNIQUEIDENTIFIER

Remarks

The NEWID function can be used in a DEFAULT clause for a column.

The NEWID function is non-deterministic; successive calls will return different values. The query optimizer does not cache the results of the NEWID function.

UUIDs can be used to uniquely identify rows in a table. A value produced on one computer does not match a value produced on another computer, so they can be used as keys in synchronization and replication environments.

UUID values are also referred to as GUID (Globally Unique Identifier) values.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a table named mytab with two columns. Column pk has a unique identifier data type, and assigns the NEWID function as the default value. Column c1 has an integer data type.

```
CREATE TABLE mytab(
   pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
   c1 INT );
```

The following statement returns a unique identifier as a string:

```
SELECT UUIDTOSTR( NEWID() );
```

For example, the value returned might be 96603324-6FF6-49DE-BF7D-F44C1C7E6856.

Related Information

Function Caching
The NEWID Default
uuid_has_hyphens Option
STRTOUUID Function [String] [page 575]
UUIDTOSTR Function [String] [page 615]
UNIQUEIDENTIFIER Data Type [page 183]

1.3.2.143 NEXT_CONNECTION Function [System]

Returns an identifying number for the next connection.

Syntax

```
NEXT_CONNECTION( connection-id [, database-id ] )
```

Parameters

connection-id

An integer, usually returned from a previous call to NEXT_CONNECTION. If `connection-id` is NULL, NEXT_CONNECTION returns the most recent connection ID.

database-id

An integer representing one of the databases on the current server. If you supply no `database-id`, the current database is used. If you supply NULL, then NEXT_CONNECTION returns the next connection regardless of database.

Returns

INT

Remarks

NEXT_CONNECTION can be used to enumerate the connections to a database. Connection IDs are generally created in monotonically increasing order. This function returns the next connection ID in reverse order.

To get the connection ID value for the most recent connection, enter NULL as the `connection-id`. To get the subsequent connection, enter the previous return value. The function returns NULL when there are no more connections in the order.

NEXT_CONNECTION is useful for disconnecting all the connections created before a specific time. However, because NEXT_CONNECTION returns the connection IDs in reverse order, connections made after the function is started are not returned. To ensure that all connections are disconnected, prevent new connections from being created before you run NEXT_CONNECTION.

Privileges

No privileges are required to execute this function to return the current connection ID. To execute this function for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns an identifier as an integer value for the first connection on the current database:

```
SELECT NEXT_CONNECTION( NULL );
```

The following statement returns an integer value like 5.

```
SELECT NEXT_CONNECTION( 10 );
```

The following call returns the next connection ID in reverse order from the specified `connection-id` on the current database:

```
SELECT NEXT_CONNECTION( connection-id );
```

The following call returns the next connection ID in reverse order from the specified `connection-id`, regardless of database:

```
SELECT NEXT_CONNECTION( connection-id, NULL );
```

The following call returns the next connection ID in reverse order from the specified `connection-id` on the specified database:

```
SELECT NEXT_CONNECTION( connection-id, database-id );
```

The following call returns the first or earliest connection, regardless of database:

```
SELECT NEXT_CONNECTION( NULL, NULL );
```

The following call returns the first or earliest connection on the specified database:

```
SELECT NEXT_CONNECTION( NULL, database-id );
```

1.3.2.144 NEXT_DATABASE Function [System]

Returns an identifying number for a database.

Syntax

```
NEXT_DATABASE( database-id )
```

Parameters

database-id

An integer that specifies the ID number of the database.

Returns

INT

Remarks

The NEXT_DATABASE function is used to enumerate the databases running on a database server. To get the first database specify NULL; to get each subsequent database, specify the previous return value. The function returns NULL when there are no more databases. The database ID numbers are not returned in a particular order, but you can tell the order in which databases were started on the server using the database ID. The first database started on the server is assigned the value 0, and for subsequent databases started on the server, the database IDs are incremented by 1.

Privileges

No privileges are required to execute this function to return the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 0, the first database value:

```
SELECT NEXT_DATABASE( NULL );
```

The following statement returns `NULL`, only when one database has been started:

```
SELECT NEXT_DATABASE( 0 );
```

Related Information

DB_NAME Function [System] [page 336]
sa_db_list System Procedure [page 1565]

1.3.2.145 NEXT_HTTP_HEADER Function [Web Service]

Returns the next HTTP header name.

Syntax

```
NEXT_HTTP_HEADER( header-name )
```

Parameters

header-name

The name of the previous request header. If header-name is NULL, this function returns the name of the first HTTP request header.

Returns

LONG VARCHAR.

Note

The result data type is a LONG VARCHAR. If you use NEXT_HTTP_HEADER in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTML_DECODE to the correct data type and size.

Remarks

This function is used to iterate over the HTTP request headers returning the next HTTP header name. Calling it with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the function. This function returns NULL when called with the name of the last header, or when not called from a web service.

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order they appear in the HTTP request.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement displays the name and values of the HTTP request headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:

```
BEGIN
  declare header_name long varchar;
  declare header_value long varchar;
  set header_name  = NULL;
header_loop:
  LOOP
    SET header_name = NEXT_HTTP_HEADER( header_name );
    IF header_name IS NULL THEN
      LEAVE header_loop
    END IF;
    SET header_value = HTTP_HEADER( header_name );
    MESSAGE 'HEADER: ', header_name, '=',
            header_value TO CONSOLE;
  END LOOP;
END;
```

Related Information

How to Access Client-supplied HTTP Variables and Headers
Web Services Functions [page 221]
HTTP_HEADER Function [Web Service] [page 412]
sa_http_header_info System Procedure [page 1613]
Web Services System Procedures [page 1512]

1.3.2.146 NEXT_HTTP_RESPONSE_HEADER Function [Web Service]

Returns the next HTTP response header name.

Syntax

```
NEXT_HTTP_RESPONSE_HEADER( header-name )
```

Parameters

header-name

The name of the previous response header. If header-name is NULL, this function returns the name of the first HTTP response header.

Returns

LONG VARCHAR

Remarks

This function is used to iterate over the HTTP response headers returning the next HTTP response header name. Calling it with NULL causes it to return the name of the first response header. Subsequent response headers are retrieved by passing the name of the previous response header to the function. This function returns NULL when called with the name of the last response header, or if it is not called from a web service.

Calling this function repeatedly returns all the response header fields exactly once, but not necessarily in the order they appear in the HTTP response.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement displays the name and values of the HTTP response headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:

```
BEGIN
  declare header_name long varchar;
  declare header_value long varchar;
  set header_name  = NULL;
header_loop:
  LOOP
    SET header_name = NEXT_HTTP_RESPONSE_HEADER( header_name );
    IF header_name IS NULL THEN
      LEAVE header_loop
    END IF;
    SET header_value = HTTP_RESPONSE_HEADER( header_name );
    MESSAGE 'RESPONSE HEADER: ', header_name, '=', header_value TO CONSOLE;
  END LOOP;
```

Related Information

HTTP Request Header Management
Web Services Functions [page 221]
HTTP_RESPONSE_HEADER Function [Web Service] [page 414]
Web Services System Procedures [page 1512]

1.3.2.147 NEXT_HTTP_VARIABLE Function [Web Service]

Returns the next HTTP variable name.

Syntax

NEXT_HTTP_VARIABLE(`var-name`*)*

Parameters

var-name

The name of the previous variable. If `var-name` is NULL, this function returns the name of the first HTTP variable.

Returns

LONG VARCHAR.

Remarks

This function iterates over the HTTP variables included within a request. Calling it with NULL causes it to return the name of the first variable. Subsequent variables are retrieved by passing the function the name of the previous variable. This function returns NULL when called with the name of the final variable or when not called from a web service.

Calling this function repeatedly returns all the variables exactly once, but not necessarily in the order they appear in the HTTP request. The variables url or url1, url2, ..., url10 are included if URL PATH is set to ON or ELEMENTS, respectively.

Standards and Compatibility

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the name of the first HTTP variable when used within a stored procedure that is called by an HTTP web service:

```
BEGIN
DECLARE variable_name LONG VARCHAR;
DECLARE variable_value LONG VARCHAR;
SET variable_name = NULL;
SET variable_name = NEXT_HTTP_VARIABLE( variable_name );
SET variable_value = HTTP_VARIABLE( variable_name );
END;
```

Related Information

How to Access Client-supplied HTTP Variables and Headers
Web Services Functions [page 221]
HTTP_VARIABLE Function [Web Service] [page 417]
NEXT_HTTP_HEADER Function [Web Service] [page 472]
sa_http_variable_info System Procedure [page 1618]

Web Services System Procedures [page 1512]

1.3.2.148 NEXT_SOAP_HEADER Function [SOAP]

Returns the next header key in a SOAP request header.

Syntax

```
NEXT_SOAP_HEADER( header-key )
```

Parameters

header-key

The XML local name of the top level XML element for the given header entry.

Returns

LONG VARCHAR

Remarks

If you specify NULL for the `header-key`, the function returns the header key for the first header entry found in the SOAP header.

This function returns NULL if called with the last `header-key`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement processes all the keys located in the SOAP request header when used within a stored procedure that is called by an HTTP web service. When it processes the Authentication key, it also obtains the key's value.

```
BEGIN
  DECLARE hd_key LONG VARCHAR;
  DECLARE hd_entry LONG VARCHAR;
header_loop:
  LOOP
    SET hd_key = NEXT_SOAP_HEADER( hd_key );
    IF hd_key IS NULL THEN
      -- no more header entries
      LEAVE header_loop;
    END IF;
    IF hd_key = 'Authentication' THEN
      SET hd_entry = SOAP_HEADER( hd_key );
    END IF;
  END LOOP header_loop;
END;
```

Related Information

Web Services Functions [page 221]
Tutorial: Using a Database Server to Access a SOAP/DISH Service
SOAP_HEADER Function [SOAP] [page 554]
Web Services System Procedures [page 1512]

1.3.2.149 NOW Function [Date and Time]

Returns the current date and time as a TIMESTAMP value. The accuracy is limited by the accuracy of the system clock.

Syntax

```
NOW( [ * ] )
```

Returns

TIMESTAMP

Remarks

NOW is equivalent to the GETDATE function and the CURRENT TIMESTAMP special value. NOW(*) and NOW() are equivalent constructions.

Each instance of the NOW function in a request is evaluated at most once. Multiple instances of NOW in the same request may or may not share the identical TIMESTAMP value.

> **i** Note
>
> If the database is using a simulated time zone, the simulated time zone is used to calculate the results of this function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the current date and time:

```
SELECT NOW( * );
```

Related Information

Time Zone Management
CURRENT TIME Special Value [page 95]
CURRENT TIMESTAMP Special Value [page 96]
CURRENT UTC TIMESTAMP Special Value [page 99]
DATE Data Type [page 168]
DATE Function [Date and Time] [page 315]
DATETIME Data Type [page 169]
DATETIME Function [Date and Time] [page 324]
DATETIMEOFFSET Data Type [page 171]
Expressions in SQL Statements [page 34]
GETDATE Function [Date and Time] [page 392]
ISDATE Function [Data Type Conversion] [page 425]
SMALLDATETIME Data Type [page 173]
TIME Data Type [page 174]

TIMESTAMP Special Value [page 111]
TIMESTAMP Data Type [page 176]
UTC TIMESTAMP Special Value [page 113]
Creating Simulated Time Zones (SQL)

1.3.2.150 NULLIF Function [Miscellaneous]

Provides an abbreviated CASE expression by comparing expressions.

Syntax

```
NULLIF( expression-1, expression-2 )
```

Parameters

expression-1

An expression to be compared.

expression-2

An expression to be compared.

Returns

Data type of the first argument.

Remarks

NULLIF compares the values of the two expressions.

If the first expression equals the second expression, NULLIF returns NULL.

If the first expression does not equal the second expression, or if the second expression is NULL, NULLIF returns the first expression.

The NULLIF function provides a short way to write some CASE expressions.

Standards

ANSI/ISO SQL Standard

Core Feature.

Example

The following statement returns the value `a`:

```
SELECT NULLIF( 'a', 'b' );
```

The following statement returns `NULL`:

```
SELECT NULLIF( 'a', 'a' );
```

Related Information

CASE Expressions [page 39]

1.3.2.151 NUMBER Function [Miscellaneous]

Generates numbers starting at 1 for each successive row in the results of the query. The NUMBER function is primarily intended for use in SELECT lists.

Syntax

```
NUMBER( [ * ] )
```

Returns

INT

Remarks

Due to limitations imposed by the NUMBER function described below, use the ROW_NUMBER function instead. The ROW_NUMBER function provides the same functionality, but without the limitations of the NUMBER function.

You can use NUMBER(*) in a SELECT list to provide a sequential numbering of the rows in the result set. NUMBER(*) returns the value of the ANSI row number of each result row. The NUMBER function can return positive or negative values, depending on how the application scrolls through the result set. For insensitive cursors, the value of NUMBER(*) will always be positive because the entire result set is materialized at OPEN.

In addition, the row number may be subject to change for some cursor types. The value is fixed for insensitive cursors and scroll cursors. If there are concurrent updates, it may change for dynamic and sensitive cursors.

A syntax error is generated if you use the NUMBER function in: a DELETE statement, a WHERE clause, a HAVING clause, an ORDER BY clause, a subquery, a query involving aggregation, any constraint, a GROUP BY clause, a DISTINCT clause, a set operator (UNION, EXCEPT, INTERSECT), or a derived table.

NUMBER(*) can be used in a view (subject to the above restrictions), but the view column corresponding to the expression involving NUMBER(*) can be referenced at most once in the query or outer view, and the view cannot participate as a NULL-supplying table in a left outer join or full outer join.

In Embedded SQL, care should be exercised when using a cursor that references a query containing a NUMBER(*) function. In particular, this function returns negative numbers when a database cursor is positioned using relative to the end of the cursor (an absolute position with a negative offset).

You can use NUMBER in the right side of an assignment in the SET clause of an UPDATE statement. For example, `SET x = NUMBER(*).`

The NUMBER function can also be used to generate primary keys when using the INSERT from SELECT statement, although using an AUTOINCREMENT clause is a preferred mechanism for generating sequential primary keys.

NUMBER(*) and NUMBER() are semantically equivalent.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns a sequentially numbered list of departments:

```
SELECT NUMBER( * ), DepartmentName
FROM GROUPO.Departments
WHERE DepartmentID > 5
ORDER BY DepartmentName;
```

Related Information

ROW_NUMBER Function [Miscellaneous] [page 537]

CREATE TABLE Statement [page 1002]
INSERT Statement [page 1232]

1.3.2.152 PATINDEX Function [String]

Returns an integer representing the starting position of the first occurrence of a pattern in a string.

Syntax

```
PATINDEX( '%pattern%', string-expression )
```

Parameters

pattern

The pattern to be searched for. If the leading percent wildcard is omitted, the PATINDEX function returns one (1) if the pattern occurs at the beginning of the string, and zero if it does not.

The pattern uses the same wildcards as the LIKE comparison. These wildcards are listed in the following table.

The pattern for UltraLite uses the wildcards in the following table:

Wildcard	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters
[]	Any single character in the specified range or set
[^]	Any single character not in the specified range or set

string-expression

The string to be searched for the pattern.

Returns

INT

Remarks

The PATINDEX function returns the starting position of the first occurrence of the pattern. If the pattern is not found, it returns zero (0).

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 2:

```
SELECT PATINDEX( '%hoco%', 'chocolate' );
```

The following statement returns the value 11:

```
SELECT PATINDEX( '%4_5_', '0a1A 2a3A 4a5A' );
```

The following statement returns 14 which is the first non-alphanumeric character in the string expression. The pattern `'%[^a-z0-9]%'` can be used instead of `'%[^a-zA-Z0-9]%'` if the database is case insensitive.

```
SELECT PATINDEX( '%[^a-zA-Z0-9]%', 'SQLAnywhere17 has many new features' );
```

The following statement can be used to retrieve everything up to and including the first non-alphanumeric character in a string:

```
SELECT LEFT( @string, PATINDEX( '%[^a-zA-Z0-9]%', @string ) );
```

The following statements create a table, myTable, and populate it with various strings containing alphanumeric characters, spaces (blanks), and non-alphanumeric characters. Then, the SELECT statement and subsequent results show how you can use PATINDEX to find the starting position of spaces and non-alphanumeric characters in the strings:

```
CREATE TABLE myTable( col1 LONG VARCHAR );
INSERT INTO myTable (col1) VALUES( 'the quick brown fox jumped over the lazy
dog' ),
( 'the quick brown fox $$$$ jumped over the lazy dog' ),
( 'the quick brown fox 0999 jumped over the lazy dog' ),
( 'the quick brown fox ** jumped over the lazy dog' ),
( 'thequickbrownfoxjumpedoverthelazydog' ),
( 'thequickbrownfoxjum999pedoverthelazydog' ),
( 'thequick$$$$brownfox' ),
( 'the quick brown fox$$ jumped over the lazy dog' );
SELECT col1,
   //position of first non-alphanumeric character or space:
   PATINDEX( '%[^a-z0-9]%', col1) AS blank_posn,
   //position of first non-alphanumeric char that isn't a space:
   PATINDEX( '%[^ a-z0-9]%', col1) AS non_alpha_char,
   //everything up to and including first non-alphanumeric char that isn't a
space:
   LEFT ( col1, PATINDEX( '%[^ a-zA-Z0-9]%', col1) ) AS left_str,
   //first non-alphanumeric char that isn't a space, and everything to the right:
   SUBSTRING ( col1, PATINDEX( '%[^ a-zA-Z0-9]%', col1) ) AS sub_str
FROM myTable;
```

col1	blank_posn	non_alpha_char	left_str	sub_str
the quick brown fox jumped over the lazy dog	4	0		the quick brown fox jumped over the lazy dog
the quick brown fox $$ $$ jumped over the lazy dog	4	21	the quick brown fox $	$$$$ jumped over the lazy dog
the quick brown fox 0999 jumped over the lazy dog	4	0		the quick brown fox 0999 jumped over the lazy dog
the quick brown fox ** jumped over the lazy dog	4	21	the quick brown fox *	** jumped over the lazy dog
thequickbrownfoxjum-pedoverthelazydog	0	0		thequickbrownfoxjum-pedoverthelazydog
thequickbrownfox-jum999pedoverthela-zydog	0	0		thequickbrownfox-jum999pedoverthela-zydog
thequick$$$$brown-fox	9	9	thequick$	$$$$brownfox
the quick brown fox$$ jumped over the lazy dog	4	20	the quick brown fox$	$$ jumped over the lazy dog

Related Information

String Functions [page 222]
LIKE Search Condition [page 67]
LOCATE Function [String] [page 442]

1.3.2.153 PERCENT_RANK Function [Ranking]

For any row X, defined by the function's arguments and ORDER BY specification, the PERCENT_RANK function determines the rank of row X - 1, divided by the number of rows in the group.

Syntax

```
PERCENT_RANK( ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Returns

The PERCENT_RANK function returns a DOUBLE value between 0 and 1.

Remarks

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause. See the `window-spec` definition for the WINDOW clause.

Standards

ANSI/ISO SQL Standard

PERCENT_RANK is part of optional ANSI/ISO SQL Language Feature T612, "Advanced OLAP operations".

Example

The following example returns a result set that shows the ranking of New York employees' salaries in descending order by gender:

```
SELECT DepartmentID, Surname, Salary, Sex,
PERCENT_RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees
WHERE State IN ('NY');
```

DepartmentID	Surname	Salary	Sex	Rank
200	Martel	55700.000	M	0
100	Guevara	42998.000	M	0.333333333
100	Soo	39075.000	M	0.666666667
400	Ahmed	34992.000	M	1
300	Davidson	57090.000	F	0
400	Blaikie	54900.000	F	0.333333333

DepartmentID	Surname	Salary	Sex	Rank
100	Whitney	45700.000	F	0.666666667
400	Wetherby	35745.000	F	1

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
CUME_DIST Function [Ranking] [page 311]
DENSE_RANK Function [Ranking] [page 346]
RANK Function [Ranking] [page 501]

1.3.2.154 PI Function [Numeric]

Returns the numeric value PI.

Syntax

```
PI( [ * ] )
```

Returns

DOUBLE

Remarks

This function returns a DOUBLE value.

PI(*) and PI() are semantically equivalent.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 3.141592653(...):

```
SELECT PI( * );
```

1.3.2.155 PLAN Function [Miscellaneous]

Returns the long plan optimization strategy of a SQL statement, as a string.

Syntax

```
PLAN( string-expression [, cursor-type [, update-status ] ]  )
```

Parameters

string-expression

The SQL statement, which is commonly a SELECT statement but which may also be an UPDATE, MERGE, or DELETE statement.

cursor-type

A string whose value can be one of asensitive (default), insensitive, sensitive, or keyset-driven.

update-status

A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor:

Value	Description
READ-ONLY	The cursor is read-only.
READ-WRITE (default)	The cursor can be read or written to.
FOR UPDATE	The cursor can be read or written to. This is exactly the same as READ-WRITE.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement passes a SELECT statement as a string parameter and returns the plan for executing the query:

```
SELECT PLAN(
   'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100' );
```

This information can help with decisions about indexes to add or how to structure your database for better performance.

The following statement returns a string containing the text plan for an INSENSITIVE cursor over the query `SELECT * FROM Departments WHERE DepartmentID > 100;`:

```
SELECT PLAN(
   'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100',
   'insensitive',
   'read-only' );
```

Related Information

Advanced: Query Execution Plans
EXPLANATION Function [Miscellaneous] [page 378]
GRAPHICAL_PLAN Function [Miscellaneous] [page 393]

1.3.2.156 POWER Function [Numeric]

Calculates one number raised to the power of another.

Syntax

```
POWER( numeric-expression-1, numeric-expression-2 )
```

Parameters

numeric-expression-1

The base.

numeric-expression-2

The exponent.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If any argument is NULL, the result is a NULL value.

Standards

ANSI/ISO SQL Standard

The POWER function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 64:

```
SELECT POWER( 2, 6 );
```

1.3.2.157 PROPERTY Function [System]

Returns the value of the specified database server property as a string.

Syntax

```
PROPERTY( { property-id | property-name } [, second-parameter ] )
```

Parameters

property-id

An integer that is the property-number of the database server property. This number can be determined from the PROPERTY_NUMBER function. The `property-id` is commonly used when looping through a set of properties.

property-name

A string giving the name of the database property.

second-parameter

You can specify a second parameter for some properties, as follows:

Property	Second parameter	Description
EventTypeDesc	`positive-integer`	Specify an event ID to return the event type description.
EventTypeName	`positive-integer`	Specify an event ID to return the event type name.
FunctionMaxParms	`positive-integer`	Specify a function number to return the maximum number of parameters that can be specified for the function.
FunctionMinParms	`positive-integer`	Specify a function number to return the minimum number of parameters that must be specified for the function.
FunctionName	`positive-integer`	Specify a function number to return the function name.
Message	`positive-integer`	Specify a line number to return the contents of the corresponding line in the database server messages window, prefixed by the date and time the message appeared.
MessageText	`positive-integer`	Specify a line number to return the text associated with the specified line number in the database server messages window, without a date and time prefix.
MessageTime	`positive-integer`	Specify a line number to return the date and time associated with the specified line number in the database server messages window.
RemoteCapability	`positive-integer`	Specify a remote capability ID to return the remote capability name associated with the ID.

Returns

VARCHAR, LONG VARCHAR

Remarks

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the name of the current database server:

```
SELECT PROPERTY( 'Name' );
```

Related Information

List of Database Server Properties
DB_PROPERTY Function [System] [page 337]

1.3.2.158 PROPERTY_DESCRIPTION Function [System]

Returns a description of a property.

Syntax

```
PROPERTY_DESCRIPTION( { property-id | property-name } )
```

Parameters

property-id

An integer that is the property-number of the database property. This number can be determined from the PROPERTY_NUMBER function. The `property-id` is commonly used when looping through a set of properties.

property-name

A string giving the name of the database property.

Returns

VARCHAR

Remarks

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns `Number of index insertions`, the description of the IndAdd property:

```
SELECT PROPERTY_DESCRIPTION( 'IndAdd' );
```

Related Information

Connection, Database, and Database Server Properties

1.3.2.159 PROPERTY_IS_TRACKABLE Function [System]

Returns whether or not you can maintain historical data for the specified database server property by storing its tracked values.

Syntax

```
PROPERTY_IS_TRACKABLE( property-ID )
```

Parameters

property-ID

The PropNum of the database server property. You can find the PropNum of the database server property by running the sa_eng_properties system procedure or by calling the PROPERTY_NUMBER function.

Returns

1 if the database server property can be tracked; otherwise, returns 0.

Remarks

Only database properties that return a numeric value can be tracked.

Example

The following example returns all database server properties that are trackable:

```
SELECT PropName FROM sa_eng_properties( ) WHERE PROPERTY_IS_TRACKABLE( PropNum )
= 1;
```

Related Information

Viewing All Trackable Database Server Property Values
sp_property_history System Procedure [page 1807]
-sf Database Server Option
sa_server_option System Procedure [page 1698]

sa_db_option System Procedure [page 1566]
-phl Database Server Option
-phs Database Server Option
List of Database Server Properties
List of Database Properties
System Privileges
sa_eng_properties System Procedure [page 1587]
PROPERTY_NUMBER Function [System] [page 496]

1.3.2.160 PROPERTY_NAME Function [System]

Returns the name of the property with the supplied property ID for the specified connection level.

Syntax

```
PROPERTY_NAME( property-id [, property-scope ] )
```

```
property-scope:
NULL
| 'server'
| 'database'
| 'db'
| 'connection'
| 'conn'
```

Parameters

property-id

The property ID of the database property.

property-scope

The scope of the property, or NULL.

Returns

VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the server-level property associated with property ID 102:

```
SELECT PROPERTY_NAME( 102, 'server' );
```

Related Information

List of Connection Properties
List of Database Server Properties

1.3.2.161 PROPERTY_NUMBER Function [System]

Returns the property number of the property with the supplied property-name.

Syntax

```
PROPERTY_NUMBER( property-name )
```

Parameters

property-name

A property name.

Returns

INT

Remarks

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property. When either property number or property name

can be used, it is preferable to use the property name. Always use the PROPERTY_NUMBER function to ensure that the property number is current for the server being used.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the property number of the PAGESIZE property as an integer:

```
SELECT PROPERTY_NUMBER( 'PAGESIZE' );
```

Related Information

Connection, Database, and Database Server Properties

1.3.2.162 QUARTER Function [Date and Time]

Returns a number indicating the quarter of the year from the supplied TIMESTAMP expression.

Syntax

```
QUARTER( timestamp-expression )
```

Parameters

timestamp-expression

The date you want the quarter for.

Returns

INTEGER

Remarks

The quarters are as follows:

Quarter	Period (inclusive)
1	January 1 to March 31
2	April 1 to June 30
3	July 1 to September 30
4	October 1 to December 31

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 2:

```
SELECT QUARTER( '1987/05/02' );
```

1.3.2.163 RADIANS Function [Numeric]

Converts a number from degrees to radians.

Syntax

```
RADIANS( numeric-expression )
```

Parameters

numeric-expression

A number, in degrees. This angle is converted to radians.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns a value of approximately 0.5236:

```
SELECT RADIANS( 30 );
```

1.3.2.164 RAND Function [Numeric]

Returns a random number in the interval 0 to 1, with an optional seed.

Syntax

```
RAND( [integer-expression] )
```

Parameters

integer-expression

An optional seed used to create a random number. This argument allows you to create repeatable random number sequences.

Returns

DOUBLE

Remarks

The RAND function is a multiplicative linear congruential random number generator. See Park and Miller (1988), CACM 31(10), pp. 1192-1201 and Press et al. (1992), Numerical Recipes in C (2nd edition, Chapter 7, pp. 279). The result of calling the RAND function is a pseudo-random number `n` where $0 < n < 1$ (neither 0.0 nor 1.0 can be the result).

When a connection is made to the server, the random number generator seeds an initial value. Each connection is uniquely seeded so that it sees a different random sequence from other connections. You can also specify a seed value (`integer-expression`) as an argument. Normally, you should only do this once before requesting a sequence of random numbers through successive calls to the RAND function. If you initialize the seed value more than once, the sequence is restarted. If you specify the same seed value, the same sequence is generated. Seed values that are close in value generate similar initial sequences, with divergence further out in the sequence.

Never combine the sequence generated from one seed value with the sequence generated from a second seed value, in an attempt to obtain statistically random results. In other words, do not reset the seed value at any time during the generation of a sequence of random values.

The RAND function is treated as a non-deterministic function. The query optimizer does not cache the results of the RAND function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements produce eleven random results. Each subsequent call to the RAND function where a seed is not specified continues to produce different results:

```
SELECT RAND( 1 );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
```

The following statement produces two sets of results with identical sequences, since the seed value is specified twice:

```
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
```

```
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
```

The following example produces five results that are near each other in value, and do not have a random distribution. For this reason, calling the RAND function more than once with similar seed values is not recommended:

```
SELECT RAND( 1 ), RAND( 2 ), RAND( 3 ), RAND( 4 ), RAND( 5 );
```

The following example produces five identical results, and should be avoided:

```
SELECT RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 );
```

Related Information

Function Caching

1.3.2.165 RANK Function [Ranking]

Calculates the value of a rank in a group of values. For ties, the RANK function leaves a gap in the ranking sequence.

Syntax

```
RANK( ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Returns

INTEGER

Remarks

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause. See the `window-spec` definition for the WINDOW clause.

Standards

ANSI/ISO SQL Standard

The RANK function is part of optional ANSI/ISO SQL Language Feature T612, "Advanced OLAP operations".

Example

The following example provides a rank in descending order of employees' salaries in Utah and New York. The 7th and 8th employees have an identical salary and therefore share the 7th place ranking. The employee that follows receives the 9th place ranking, which leaves a gap in the ranking sequence (no 8th place ranking).

```
SELECT Surname, Salary, State,
RANK() OVER (ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees WHERE State IN ('NY','UT');
```

Surname	Salary	State	Rank
Shishov	72995.000	UT	1
Wang	68400.000	UT	2
Cobb	62000.000	UT	3
Morris	61300.000	UT	4
Davidson	57090.000	NY	5
Martel	55700.000	NY	6
Blaikie	54900.000	NY	7
Diaz	54900.000	NY	7
Driscoll	48023.690	UT	9
Hildebrand	45829.000	UT	10
Whitney	45700.000	NY	11
...	...	...	...
Lynch	24903.000	UT	19

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
CUME_DIST Function [Ranking] [page 311]
DENSE_RANK Function [Ranking] [page 346]

ROW_NUMBER Function [Miscellaneous] [page 537]
PERCENT_RANK Function [Ranking] [page 485]

1.3.2.166 READ_CLIENT_FILE Function [String]

Reads data from the specified file on the client computer.

Syntax

```
READ_CLIENT_FILE( client-filename-expression )
```

Parameters

client-filename-expression

CHAR value indicating the name of the file on the client computer. The path is resolved on the client computer relative to the current working directory of the client application.

Returns

LONG BINARY

Remarks

The value returned by the READ_CLIENT_FILE function represents the contents of the specified client file. You can use the function in syntax wherever a BINARY expression is allowed.

Since the data returns as a binary string, if the data is in another character set, or is compressed, or is encrypted, you may also need to perform character set conversion, decompression, or decryption on it.

During evaluation of READ_CLIENT_FILE, the database server initiates the transfer of the specified file from the client. The client, upon receiving the transfer request, obtains a shared lock on the client file, and holds the lock until the database server requests the client to terminate the request.

Reading of the file is performed by the client software library, and the transfer of data is done by using the **Command Sequence** (**CmdSeq**) communication protocol. Client-side data transfers are not supported by the **Tabular Data Stream** (**TDS**) protocol.

Privileges

When reading from a file on a client computer:

- You must have the READ CLIENT FILE system privilege.
- Read permissions are required on the directory being read from.
- The allow_read_client_file database option must be enabled.
- The READ_CLIENT_FILE feature must be enabled (-sf server option).

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Access to Data on Client Computers
-sf Database Server Option
allow_read_client_file Option
DECOMPRESS Function [String] [page 339]
DECRYPT Function [String] [page 340]
CSCONVERT Function [String] [page 309]

1.3.2.167 READ_SERVER_FILE Function [String]

Reads data from the specified file on the server and returns the full or partial contents of the file as a LONG BINARY value.

Syntax

```
READ_SERVER_FILE( filename [ , start [ , length ] ] )
```

Parameters

filename

LONG VARCHAR value indicating the path and name of the file on the server.

start

The start position of the file to read, in bytes. The first byte in the file is at position 1. A negative starting position specifies the number of bytes from the end of the file rather than from the beginning.

- If `start` is not specified, a value of 1 is used.
- If `start` is zero and `length` is non-negative, a value of 1 is used.
- If `start` is zero and `length` is negative, a value of -1 is used.

length

The length of the file to read, in bytes.

- If `length` is not specified, the function reads from the starting position to the end of the file.
- If `length` is positive, the function reads at most `length` bytes beginning at the starting position.
- If `length` is negative, the function returns at most `length` ending at the starting position.

Returns

LONG BINARY

Remarks

This function returns the full or partial (if `start` and/or `length` are specified) contents of the named file as a LONG BINARY value. If the file does not exist or cannot be read, NULL is returned.

`filename` is relative to the starting directory of the database server.

The READ_SERVER_FILE function supports reading files larger than 2GB. However, the returned content is limited to 2GB. If the returned content exceeds this limit, a SQL error is returned.

If the data file is in a different character set, you can use the CSCONVERT function to convert it. You can also use the CSCONVERT function to address the character set conversion requirements you may have when using the READ_SERVER_FILE server function.

If disk sandboxing is enabled, the file referenced in `filename` must be in an accessible location.

Privileges

When reading from a file on a client computer:

- You must have the READ FILE system privilege.
- You must have read permissions on the directory being read from.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement reads 20 bytes in a file, starting from byte 100 of the file.

```
SELECT READ_SERVER_FILE( 'c:\\data.txt', 100, 20 )
```

Related Information

Disk Sandboxing
xp_read_file System Procedure [page 1871]
CSCONVERT Function [String] [page 309]

1.3.2.168 REGEXP_SUBSTR Function [String]

Extracts substrings from strings using regular expressions.

Syntax

```
REGEXP_SUBSTR( expression
 , regular-expression
[ , start-offset
[ , occurrence-number
[ , escape-expression ] ] ]
)
```

Parameters

expression

The string to be searched.

regular-expression

The pattern you are trying to match.

start-offset

The offset into `expression` at which to start searching. `start-offset` is expressed as a positive integer, and reflects the number of characters to count when starting from the left side of the string. The default is 1 (the start of the string).

occurrence-number

For multiple matches within `expression`, specify an integer indicating the occurrence to locate. For example, 3 finds the third occurrence. The default is 1.

escape-expression

The escape character to use for `regular-expression`. The default is the backslash character (\).

Returns

LONG VARCHAR

Remarks

REGEXP_SUBSTR returns NULL if `regular-expression` is not found.

Similar to the REGEXP search condition, the REGEXP_SUBSTR function uses code points for matching and range evaluation. Database case sensitivity does not impact results.

When matching against a character class that contains only a sub-character class, include the outer square brackets and the square brackets for the sub-character class (for example, `REGEXP_SUBSTR (expression, '[[:digit:]]')`).

If `start-offset` is specified, that offset specifies the start of the expression to be matched. In particular, ^ matches the beginning of the expression starting at `start-offset`.

Standards

ANSI/ISO SQL Standard

Not in the standard. The corresponding function in the SQL standard is the SUBSTRING_REGEX function, which has similar parameters. SUBSTRING_REGEX is part of optional Language Feature F844.

Example

The following statement breaks values in the Employees.Street column into street number and street name:

```
SELECT REGEXP_SUBSTR( Street, '^\S+' ) as street_num,
 REGEXP_SUBSTR( Street, '(?<=^\S+\s+).*$' ) AS street_name
    FROM GROUPO.Employees;
```

street_num	street_name
9	East Washington Street
7	Pleasant Street
539	Pond Street
1244	Great Plain Avenue
...	...

To determine whether the IP address of the current connection is in a range of IP addresses (in this case, 10.25.101.xxx or 10.25.102.xxx), you can execute the following statement:

```
IF REGEXP_SUBSTR( CONNECTION_PROPERTY( 'NodeAddress' ), '\\d+\\.\\d+\\.\\d+' )
   IN ( '10.25.101' , '10.25.102' ) THEN
      MESSAGE 'In range' TO CLIENT;
ELSE
      MESSAGE 'Out of range' TO CLIENT;
END IF;
```

Related Information

LIKE, REGEXP, and SIMILAR TO Search Conditions [page 65]
Regular Expressions Overview [page 40]
Regular Expressions Syntax [page 41]
Regular Expressions: Special Sub-character Classes [page 46]
REGEXP Search Condition [page 72]

1.3.2.169 REGR_AVGX Function [Aggregate]

Computes the average of the independent variable of the regression line.

Syntax

Expression

```
REGR_AVGX( dependent-expression , independent-expression )
```

Window function

```
REGR_AVGX( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where `x` represents the `independent-expression`:

```
AVG( x )
```

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_AVGX is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement calculates the average of the dependent variable, employee age:

```
SELECT REGR_AVGX( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
AVG Function [Aggregate] [page 254]

1.3.2.170 REGR_AVGY Function [Aggregate]

Computes the average of the dependent variable of the regression line.

Syntax

Expression

```
REGR_AVGY( dependent-expression , independent-expression )
```

Window function

```
REGR_AVGY( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where `y` represents the `dependent-expression`:

```
AVG( y )
```

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_AVGY is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement calculates the average of the independent variable, employee salary:

```
SELECT REGR_AVGY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
AVG Function [Aggregate] [page 254]

1.3.2.171 REGR_COUNT Function [Aggregate]

Returns an integer that represents the number of non-NULL number pairs used to fit the regression line.

Syntax

Expression

```
REGR_COUNT( dependent-expression , independent-expression )
```

Window function

```
REGR_COUNT( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

INTEGER

Remarks

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_COUNT is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the number of non-NULL pairs that were used to fit the regression line:

```
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
COUNT Function [Aggregate] [page 300]
AVG Function [Aggregate] [page 254]
SUM Function [Aggregate] [page 581]

1.3.2.172 REGR_INTERCEPT Function [Aggregate]

Computes the y-intercept of the linear regression line that best fits the dependent and independent variables.

Syntax

Expression

```
REGR_INTERCEPT( dependent-expression , independent-expression )
```

Window function

```
REGR_INTERCEPT( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where `y` represents the `dependent-expression` and `x` represents the `independent-expression`:

```
AVG( y ) - REGR_SLOPE( y, x ) * AVG( x )
```

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_INTERCEPT is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the y-intercept of the linear regression line:

```
SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]

AVG Function [Aggregate] [page 254]

1.3.2.173 REGR_R2 Function [Aggregate]

Computes the coefficient of determination (also referred to as *R-squared* or the *goodness of fit* statistic) for the regression line.

Syntax

Expression

```
REGR_R2( dependent-expression , independent-expression )
```

Window function

```
REGR_R2( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_R2 is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the coefficient of determination for the regression line:

```
SELECT REGR_R2( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]

1.3.2.174 REGR_SLOPE Function [Aggregate]

Computes the slope of the linear regression line fitted to non-NULL pairs.

Syntax

Expression

```
REGR_SLOPE( dependent-expression , independent-expression )
```

Window function

```
REGR_SLOPE( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where `y` represents the `dependent-expression` and `x` represents the `independent-expression`:

```
COVAR_POP( y, x ) / VAR_POP( x )
```

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_SLOPE is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 935.3429749445614:

```
SELECT REGR_SLOPE( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
COVAR_POP Function [Aggregate] [page 305]
VAR_POP Function [Aggregate] [page 616]

1.3.2.175 REGR_SXX Function [Aggregate]

Returns the sum of squares of the independent expressions used in a linear regression model. The REGR_SXX function can be used to evaluate the statistical validity of a regression model.

Syntax

Expression

```
REGR_SXX( dependent-expression , independent-expression )
```

Window function

```
REGR_SXX( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where `y` represents the `dependent-expression` and `x` represents the `independent-expression`:

```
REGR_COUNT( y, x ) * VAR_POP( x )
```

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_SXX is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 5916.4800000000105:

```
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
VAR_POP Function [Aggregate] [page 616]

1.3.2.176 REGR_SXY Function [Aggregate]

Returns the sum of products of the dependent and independent variables. The REGR_SXY function can be used to evaluate the statistical validity of a regression model.

Syntax

Expression

```
REGR_SXY( dependent-expression , independent-expression )
```

Window function

```
REGR_SXY( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where `y` represents the `dependent-expression` and `x` represents the `independent-expression`:

```
REGR_COUNT( y, x ) * COVAR_POP( y, x )
```

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_SXY is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the sum of products of the dependent and independent variables:

```
SELECT REGR_SXY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]

1.3.2.177 REGR_SYY Function [Aggregate]

Returns values that can evaluate the statistical validity of a regression model.

Syntax

Expression

```
REGR_SYY( dependent-expression , independent-expression )
```

Window function

```
REGR_SYY( dependent-expression , independent-expression )
OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

dependent-expression

The variable that is affected by the independent variable.

independent-expression

The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (`dependent-expression` and `independent-expression`) pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where `y` represents the `dependent-expression` and `x` represents the `independent-expression`:

```
REGR_COUNT( y, x ) * VAR_POP( y )
```

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

REGR_SYY is part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 26, 708, 672,843.3002:

```
SELECT REGR_SYY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]

1.3.2.178 REMAINDER Function [Numeric]

Returns the remainder when one whole number is divided by another.

Syntax

```
REMAINDER( dividend, divisor )
```

Parameters

dividend

The dividend, or numerator of the division.

divisor

The divisor, or denominator of the division.

Returns

- INTEGER
- NUMERIC

Remarks

You can also use the MOD function to return the remainder.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 2:

```
SELECT REMAINDER( 5, 3 );
```

Related Information

MOD Function [Numeric] [page 461]

1.3.2.179 REPEAT Function [String]

Concatenates a string a specified number of times.

Syntax

```
REPEAT( string-expression, integer-expression )
```

Parameters

string-expression

The string to be repeated.

integer-expression

The number of times the string is to be repeated. If `integer-expression` is negative, an empty string is returned.

Returns

LONG VARCHAR
LONG NVARCHAR
UltraLite returns LONG VARCHAR

Remarks

If the actual length of the result string exceeds the maximum for the return type, an error occurs. The result is truncated to the maximum string size allowed.

The behavior of this function is identical to that of the REPLICATE function.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value `repeatrepeatrepeat`:

```
SELECT REPEAT( 'repeat', 3 );
```

Related Information

String Functions [page 222]
REPLICATE Function [String] [page 527]

1.3.2.180 REPLACE Function [String]

Replaces a string with another string, and returns the new results.

Syntax

```
REPLACE( original-string, search-string, replace-string )
```

Parameters

If any argument is NULL, the function returns NULL.

original-string

The string to be searched. This can be any length.

search-string

The string to be searched for and replaced with `replace-string`. This string is limited to 255 bytes. If `search-string` is an empty string, the original string is returned unchanged.

replace-string

The replacement string, which replaces `search-string`. This can be any length. If `replace-string` is an empty string, all occurrences of `search-string` are deleted.

Returns

LONG BINARY
LONG VARCHAR
LONG NVARCHAR
UltraLite does not return NVARCHAR

Remarks

This function replaces all occurrences of search-string with replace-string.

If all arguments are of binary data type, the REPLACE function behaves the same as the BYTE_REPLACE function.

Comparisons are case-sensitive on case-sensitive databases.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value `xx.def.xx.ghi`:

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' );
```

The following statement generates a result set containing ALTER PROCEDURE statements which, when executed, would repair stored procedures that reference a table that has been renamed. (To be useful, the table name must be unique.)

```
SELECT REPLACE(
   REPLACE( proc_defn, 'OldTableName', 'NewTableName' ),
   'CREATE PROCEDURE',
   'ALTER PROCEDURE')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%';
```

Related Information

String Functions [page 222]
SUBSTRING Function [String] [page 578]
CHARINDEX Function [String] [page 282]

1.3.2.181 REPLICATE Function [String]

Concatenates a string a specified number of times.

Syntax

```
REPLICATE( string-expression, integer-expression )
```

Parameters

string-expression

The string to be repeated.

integer-expression

The number of times the string is to be repeated.

Returns

LONG VARCHAR
LONG NVARCHAR
UltraLite does not return NVARCHAR

Remarks

If the actual length of the result string exceeds the maximum for the return type, an error occurs. The result is truncated to the maximum string size allowed.

The behavior of this function is identical to that of the REPEAT function.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value `repeatrepeatrepeat`:

```
SELECT REPLICATE( 'repeat', 3 );
```

Related Information

String Functions [page 222]
REPEAT Function [String] [page 524]

1.3.2.182 REVERSE Function [String]

Returns the reverse of a character expression.

Syntax

```
REVERSE( string-expression )
```

Parameters

string-expression

The string to be reversed.

Returns

- LONG VARCHAR
- LONG NVARCHAR

Remarks

This function supports NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value `cba`:

```
SELECT REVERSE( 'abc' );
```

Related Information

String Functions [page 222]

1.3.2.183 REWRITE Function [Miscellaneous]

Returns a rewritten SELECT, UPDATE, or DELETE statement.

Syntax

```
REWRITE( select-statement [ , 'ANSI' ] )
```

Parameters

select-statement

The SQL statement to which the rewrite optimizations are applied to generate the function's results.

Returns

LONG VARCHAR

Remarks

You can use the REWRITE function without the ANSI argument to help understand how the optimizer generated the access plan for a given query. In particular, you can find how the database server has rewritten the conditions in the statement's WHERE, ON, and HAVING clauses, and then determine if applicable indexes exist that can be exploited to improve the request execution time.

The statement that is returned by REWRITE may not match the semantics of the original statement. This is because several rewrite optimizations introduce internal mechanisms that cannot be translated directly into SQL. For example, the server's use of row identifiers to perform duplicate elimination cannot be translated into SQL.

The rewritten query from the REWRITE function is not intended to be executable. It is a tool for analyzing performance issues by showing what gets passed to the optimizer after the rewrite phase.

There are some rewrite optimizations that are not reflected in the output of REWRITE. They include LIKE optimization, optimization for minimum or maximum functions, upper/lower elimination, and predicate subsumption.

If ANSI is specified, REWRITE returns the ANSI equivalent to the statement. In this case, only the following rewrite optimizations are applied:

- Transact-SQL outer joins are rewritten as ANSI SQL outer joins.
- Duplicate correlation names are eliminated.
- KEY and NATURAL joins are rewritten as ANSI SQL joins.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In the following statement, two rewrite optimizations are performed on a query. The first is the un-nesting of the subquery into a join between the Employees and SalesOrders tables. The second optimization simplifies the query by eliminating the primary key - foreign key join between Employees and SalesOrders. Part of this rewrite optimization is to replace the join predicate e.EmployeeID=s.SalesRepresentative with the predicate s.SalesRepresentative IS NOT NULL.

```
SELECT REWRITE( 'SELECT s.ID, s.OrderDate
    FROM GROUPO.SalesOrders s
    WHERE EXISTS ( SELECT *
       FROM GROUPO.Employees e
          WHERE e.EmployeeID = s.SalesRepresentative)' ) FROM SYS.DUMMY;
```

The query returns a single column result set containing the rewritten query with line separators.

```
'select s.ID,s.OrderDate
  from GROUPO.SalesOrders as s'
```

Semantic analysis of the query permits the removal of the WHERE clause (SalesRepresentative has a FOREIGN KEY constraint on the Employees table).

The next REWRITE statement uses the ANSI argument. This query requires that the tsql_outer_joins option be set.

```
SET TEMPORARY OPTION tsql_outer_joins = 'On';
SELECT REWRITE( 'SELECT DISTINCT s.ID, s.OrderDate, e.GivenName, e.EmployeeID
    FROM GROUPO.SalesOrders s, GROUPO.Employees e
          WHERE e.EmployeeID *= s.SalesRepresentative', 'ANSI' ) FROM SYS.DUMMY;
```

The result is the ANSI equivalent of the statement. In this case, the Transact-SQL outer join is converted to an ANSI outer join. The query returns a single column result set containing the rewritten statement with line separators.

```
'select distinct s.ID,s.OrderDate,e.GivenName,e.EmployeeID
  from GROUPO.Employees as e left outer join GROUPO.SalesOrders as s
    on e.EmployeeID = s.SalesRepresentative'
```

Related Information

Optimizations Performed During Query Processing
Transact-SQL Outer Joins (*= or =*)
Key Joins
Natural Joins
Duplicate Correlation Names in Joins (Star Joins)
extended_join_syntax Option

1.3.2.184 RIGHT Function [String]

Returns the rightmost characters of a string.

Syntax

```
RIGHT( string-expression, integer-expression )
```

Parameters

string-expression

The string to return the rightmost characters for.

integer-expression

The number of characters at the end of the string to return.

Returns

LONG VARCHAR
LONG NVARCHAR
UltraLite does not return NVARCHAR

Remarks

If the string contains multibyte characters, the number of bytes returned may be greater than the specified number of characters.

You can specify an `integer-expression` that is larger than the value in the column. In this case, the entire value is returned.

This function supports NCHAR inputs and/or outputs. Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the last 5 characters of each Surname value in the Customers table:

```
SELECT RIGHT( Surname, 5 ) FROM GROUPO.Customers;
```

Related Information

International Languages and Character Sets
String Functions [page 222]
LEFT Function [String] [page 435]

1.3.2.185 ROUND Function [Numeric]

Rounds the `numeric-expression` to the specified `integer-expression` amount of places after the decimal point.

Syntax

```
ROUND( numeric-expression, integer-expression )
```

Parameters

numeric-expression

The number, passed into the function, to be rounded.

integer-expression

A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Returns

NUMERIC

Remarks

The result of this function is either numeric or double. When there is a numeric result and the integer `integer-expression` is a negative value, the precision is increased by one.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 );
```

Related Information

TRUNCNUM Function [Numeric] [page 604]

1.3.2.186 ROW Constructor [Composite]

Returns a sequence of (`field name data type`, ...) pairs named **fields**.

Syntax

```
ROW( [expression [, expression ... ] | single-row-query-expression ] )
```

Parameters

expression

An expression representing a single field.

single-row-query-expression

A query statement that returns a single row with one or more columns used to construct fields.

Returns

A SQL ROW object.

Remarks

When the ROW constructor list contains no elements (for example, ROW()), then each element in the SQL ROW variable that you are initializing is set to NULL. Otherwise, the ROW constructor list must contain the same number of elements as the SQL ROW variable that you are initializing. In this case, the elements can be NULL or some appropriate value (for example, ROW(NULL,123)). If any field in the ROW variable is an ARRAY, it is initialized as a zero-length array.

ROW variables cannot be specified in the outermost SELECT list of a view definition, or in a top-level SELECT block or query expression that is returned to the client. ROW types cannot be stored as columns in a base or temporary table.

A ROW variable can contain arbitrary levels of nesting, resulting in complex structured types. Each ROW sub-type is given a name, and these names can be referenced by using dot-notation to select particular values.

Dot-notation permits other expressions in the same or nested query block to refer to all or portions of a ROW variable for comparison or initialization of other ROW variables.

When referring to fields within columns with a qualified name, or when referring to nested fields, surround column names with parentheses. This is illustrated in the first example below.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In the following example, fields and sub-fields in the ROW variable are initialized to NULL.

```
CREATE OR REPLACE VARIABLE myrowvar ROW ( field1 INT, field2 ROW ( id INT, name
CHAR(10) ) );
SET myrowvar = ROW();
SELECT (myrowvar).field1 AS field1, (myrowvar).field2.id AS id,
(myrowvar).field2.name AS name;
```

In the following example, fields and sub-fields in the ROW variable are initialized to the specified values. Two ROW constructors are used and one is nested inside the other.

```
CREATE OR REPLACE VARIABLE myrowvar ROW ( field1 INT, field2 ROW ( id INT, name
CHAR(10) ) );
SET myrowvar = ROW( 1, ROW( NULL, 'Jack' ) );
SELECT (myrowvar).field1 AS field1, (myrowvar).field2.id AS id,
(myrowvar).field2.name AS name;
```

The following statement illustrates how to construct ROW objects with fields initialized from columns of the same name in the Products table. The ID component of the ROW object is included in the result set.

```
SELECT ROW( ID, Name, Description ).ID as ProductID FROM GROUPO.Products;
```

The ROW object has fields named ID, Name, and Description which match column names from the Products table and which are used to extract the corresponding column data.

The following statement illustrates how to use the CAST function to assign explicit names to an implicit ROW type, and then extract the ProductID field from the casted ROW:

```
SELECT CAST( ROW( 303, 'Tee Shirt', 'Soft white cotton' )
    AS ROW( ProductID INTEGER, ProductName CHAR(25), ProductDescription
CHAR(35) )
).ProductID AS ProductID;
```

SQL expressions that use this query result as input can reference the components of the ROW object by dotted expressions on their names. ROW objects can also be constructed with a single-row query expression.

The following example illustrates how to construct a ROW object from a single-row query expression and then extract the fields from it:

```
CREATE OR REPLACE VARIABLE myrowvar ROW ( name VARCHAR(30), number INT );
SET myrowvar = ROW( SELECT Name, Quantity FROM Products WHERE ID = 300 );
SELECT (myrowvar).name AS Name, (myrowvar).number AS Quantity;
```

The following statement illustrates how to set the last_name field in the my_student ROW variable:

```
CREATE VARIABLE my_student ROW( first_name LONG VARCHAR, last_name LONG
VARCHAR );
SET student.last_name = 'Johnson';
```

The following example shows how to refer to fields within columns with a qualified name:

```
SELECT ( myderivedtable.myrowcolumn ).id AS ID
    FROM ( SELECT ROW( id, name )
           FROM GROUPO.Products
         ) AS myderivedtable( myrowcolumn );
```

Related Information

ROW and ARRAY Composite Data Types [page 185]
Comparisons of Composite Types [page 198]
CAST Function [Data Type Conversion] [page 275]

1.3.2.187 ROW_NUMBER Function [Miscellaneous]

Assigns a unique number to each row. Use this function instead of the NUMBER function.

Syntax

```
ROW_NUMBER( ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Returns

INTEGER

Remarks

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause. See the `window-spec` definition for the WINDOW clause.

Standards

ANSI/ISO SQL Standard

ROW_NUMBER is part of optional ANSI/ISO SQL Language Feature T611, "Elementary OLAP operations".

Example

The following statement returns a result set that provides unique row numbers for each employee in New York and Utah. Because the query is ordered by Salary in descending order, the first row number is given to the

employee with the highest salary in the data set. Although two employees have identical salaries, the tie is not resolved because the two employees are assigned unique row numbers.

```
SELECT Surname, Salary, State,
ROW_NUMBER() OVER (ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees WHERE State IN ('NY','UT');
```

Surname	Salary	State	Rank
Shishov	72995.000	UT	1
Wang	68400.000	UT	2
Cobb	62000.000	UT	3
Morris	61300.000	UT	4
Davidson	57090.000	NY	5
Martel	55700.000	NY	6
Blaikie	54900.000	NY	7
Diaz	54900.000	NY	8
Driscoll	48023.690	UT	9
Hildebrand	45829.000	UT	10
...	...	...	...
Lynch	24903.000	UT	19

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
NUMBER Function [Miscellaneous] [page 481]
RANK Function [Ranking] [page 501]
ROWID Function [Miscellaneous] [page 538]

1.3.2.188 ROWID Function [Miscellaneous]

Returns an UNSIGNED BIGINT value that uniquely identifies a row within a table.

Syntax

```
ROWID( correlation-name )
```

Parameters

correlation-name

The correlation name of a table used in the query. The correlation name should refer to a base table, a temporary table, a global temporary table or a proxy table (permitted only when the underlying proxy server supports a similar function). The argument of the ROWID function should not refer to a view, derived table, common table expression or a procedure.

Returns

UNSIGNED BIGINT

Remarks

Returns the row identifier of the row in the table corresponding to the given correlation name.

The value returned by the function is not necessarily constant between queries as various operations performed on the database may result in changes to the row identifiers of a table. In particular, the REORGANIZE TABLE statement is likely to result in changes to row identifiers. Additionally, row identifiers may be reused after a row has been deleted. So, users should refrain from using the ROWID function in ordinary situations; retrieval by primary key value should be used instead. It is recommended that ROWID be used only in diagnostic situations.

Although the result of this function is an UNSIGNED BIGINT, the results of most arithmetic operations on this value have no particular meaning. For example, you should not expect that adding one to a row identifier will give you the row identifier of the next row. Also, only equality and IN predicates are sargable if they involve the use of ROWID. If necessary, predicates involving ROWID, such as `ROWID( T ) = literal`, may be used to cast to a 64-bit UNSIGNED INTEGER value. If the conversion cannot be performed a data exception will occur. If the value of `literal` is an invalid row identifier then the comparison predicate evaluates to FALSE.

The ROWID function cannot be used inside a CHECK constraint on either a table or a column, nor can it be used in the COMPUTE expression for a computed column.

When used with the OPENSTRING function, the ROWID value is the row number in the input string.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the row identifier of the row in Employee where id is equal to 105:

```
SELECT ROWID( Employees ) FROM GROUPO.Employees WHERE Employees.EmployeeID = 105;
```

The following statement returns a list of the locks on rows in the Employees table along with the contents of those rows:

```
SELECT *
  FROM sa_locks() S JOIN GROUPO.Employees WITH( NOLOCK )
     ON ROWID( Employees ) = S.row_identifier
  WHERE S.table_name = 'Employees';
```

Related Information

ROW_NUMBER Function [Miscellaneous] [page 537]

1.3.2.189 RTRIM Function [String]

Removes trailing blanks or specified characters from the string.

Syntax

```
RTRIM( string-expression [ , trim-char-set ] )
```

Parameters

string-expression

The string to be trimmed.

trim-char-set

The set of characters to trim.

Returns

VARCHAR
NVARCHAR
LONG VARCHAR

LONG NVARCHAR
UltraLite VARCHAR and LONG VARCHAR

Remarks

By default, `trim-char-set` is the space character. You can specify the set of characters to be trimmed.

The actual length of the result is the length of the expression minus the number of characters removed. If all the characters are removed, the result is an empty string.

If the argument is null, the result is the NULL value.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs and `trim-char-set`.

Standards

ANSI/ISO SQL Standard

Not in the standard.

The TRIM specifications defined by the ANSI/ISO SQL Standard (LEADING and TRAILING) are supplied by the LTRIM and RTRIM functions, respectively, that are provided in the software.

Example

The following statement returns the string `Test Message`, with all trailing blanks removed:

```
SELECT RTRIM( 'Test Message     ' );
```

The following statement returns the value def after the specified trailing characters are removed:

```
SELECT RTRIM('defabcabccba', 'abc' );
```

Related Information

String Functions [page 222]
TRIM Function [String] [page 601]
LTRIM Function [String] [page 448]

1.3.2.190 SECOND Function [Date and Time]

Returns the seconds value of the TIMESTAMP argument.

Syntax

```
SECOND( timestamp-expression )
```

Parameters

timestamp-expression

The TIMESTAMP value.

Returns

SMALLINT

Remarks

Returns a number from 0 to 59 corresponding to the seconds component of the given TIMESTAMP argument value.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 25:

```
SELECT SECOND( '1998-07-13 21:21:25' );
```

1.3.2.191 SECONDS Function [Date and Time]

Manipulates a TIMESTAMP or returns the number of second boundaries between two TIMESTAMP values.

Syntax

Return the number of seconds between midnight 0000-02-29 and a TIMESTAMP value

```
SECONDS( timestamp-expression )
```

Return the number of seconds between two TIMESTAMP values

```
SECONDS( timestamp-expression, timestamp-expression )
```

Add seconds to a TIMESTAMP value

```
SECONDS( time-or-timestamp-expression, integer-expression )
```

Parameters

timestamp-expression

A TIMESTAMP value.

time-or-timestamp-expression

A value of type TIME or TIMESTAMP.

integer-expression

The number of seconds to be added to the `time-or-timestamp-expression`. If `integer-expression` is negative, the appropriate number of seconds is subtracted from `time-or-timestamp-expression`. If you supply an integer expression, the `time-or-timestamp-expression` must be explicitly cast as a TIME, DATE, or TIMESTAMP data type. If `time-or-timestamp-expression` is a DATE type, its time portion is assumed to be midnight.

Returns

UNSIGNED BIGINT when returning the number of seconds between midnight 0000-02-29 and a TIMESTAMP value

SIGNED BIGINT when returning the number of seconds between two TIMESTAMP values.

TIME or TIMESTAMP when adding seconds to a TIMESTAMP value.

Remarks

The result of the SECONDS function depends on its arguments.

Return the number of seconds between midnight 0000-02-29 and a TIMESTAMP value

If you pass a single `timestamp-expression` to the SECONDS function, it will return the number of second boundaries between midnight 0000-02-29 and `timestamp-expression` as an UNSIGNED BIGINT.

> **i** Note
>
> 0000-02 is not meant to imply an actual date; it is the default date used by the SECONDS function.

Return the number of seconds between two TIMESTAMP values

If you pass two TIMESTAMP values to the SECONDS function, the function returns the integer number of second boundaries between them as a SIGNED BIGINT value.

Add seconds to a TIMESTAMP value

If you pass a TIMESTAMP value and a INTEGER value to the SECONDS function, the function returns the TIMESTAMP result of adding the integer number of seconds to `time-or-timestamp-expression`. Similarly, if you pass a TIME value to the SECONDS function, the function returns a value of type TIME.

You can also use the DATEDIFF and DATEADD functions to perform some of these calculations.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns identical values 14400, signifying that the second TIMESTAMP value is 14400 seconds after the first:

```
SELECT
  SECONDS( '1999-07-13 06:07:12',
           '1999-07-13 10:07:12' ),
  DATEDIFF( second,
           '1999-07-13 06:07:12',
           '1999-07-13 10:07:12' );
```

The following statement returns the value 63062431632:

```
SELECT SECONDS( '1998-07-13 06:07:12' );
```

The following statements return the TIMESTAMP value 1999-05-12 21:05:12.000:

```
SELECT SECONDS( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5);
SELECT DATEADD( second, 5, '1999-05-12 21:05:07' );
```

Related Information

CAST Function [Data Type Conversion] [page 275]
DATEADD Function [Date and Time] [page 316]
DATEDIFF Function [Date and Time] [page 317]

1.3.2.192 SECURE_SIGN_MESSAGE Function [String]

Digitally signs a message.

Syntax

```
SECURE_SIGN_MESSAGE(
message
, key
[, hash-algorithm ]
)
```

```
hash-algorithm:
'SHA-1'
| 'SHA-256'
| 'MD5'
```

Parameters

message A long binary value specifying the message to be signed.
key The key to be used, in PEM format. Messages are signed by a private key and verified by a public key.
hash-algorithm (Optional). Specifies the hashing algorithm used in the signing process. 'SHA-1' is the default.

Returns

LONG BINARY

Remarks

This function creates a cryptographic hash of a message and then encrypts the hash by using the sender's private key.

Example

1. Create an RSA key pair and publish the public key so that recipients of your message can access it.

```
CREATE OR REPLACE VARIABLE @public_key LONG VARCHAR;
CREATE OR REPLACE VARIABLE @private_key LONG VARCHAR;
CALL sp_generate_key_pair( 2048, @public_key, @private_key );
```

2. The message sender executes the following statements to create a signature and to sign a message using the private key:

```
CREATE OR REPLACE VARIABLE @signature LONG BINARY;
SELECT SECURE_SIGN_MESSAGE( 'This is exactly what I said', @private_key )
INTO @signature;
```

3. The message sender sends the document, along with the signature and public key, to the message recipients.
4. The message recipients execute the following statements to verify that the message is authentic and that it has not been modified since being sent:

```
CREATE OR REPLACE VARIABLE @verified INTEGER;
SELECT SECURE_VERIFY_MESSAGE( 'This is exactly what I said', @signature,
@public_key ) INTO @verified;
```

   If the @verified variable equals 1, then the message is authentic.

Related Information

Signing and Verifying Messages
SECURE_VERIFY_MESSAGE Function [String] [page 546]
sp_generate_key_pair System Procedure [page 1777]

1.3.2.193 SECURE_VERIFY_MESSAGE Function [String]

Digitally verifies a message.

Syntax

```
SECURE_VERIFY_MESSAGE(
string-expression
, signature
, key
[, hash-algorithm ]
)
```

```
hash-algorithm:
'SHA-1'
| 'SHA-256'
| 'MD5'
```

Parameters

string-expression The message to be verified.
signature The signature to verify.
key The key to be used, in PEM format. Messages are signed by a private key and verified by a public key.
hash-algorithm (Optional). Specifies the hashing algorithm used during the verification process. 'SHA-1' is the default.

Returns

Returns 1 if the message and signature were successfully verified; otherwise 0.

Remarks

This function verifies a digital signature by decrypting the signature using the sender's key, computing the hash of the message, and then comparing the two hashes. If the two hashes match, then the recipient can confirm that the message was sent by someone with access to the sender's private key and that the message has not been modified.

Example

1. Create an RSA key pair and publish the public key so that recipients of your message can access it.

   ```
   CREATE OR REPLACE VARIABLE @public_key LONG VARCHAR;
   CREATE OR REPLACE VARIABLE @private_key LONG VARCHAR;
   CALL sp_generate_key_pair( 2048, @public_key, @private_key );
   ```

2. The message sender executes the following statements to create a signature and to sign a message using the private key:

   ```
   CREATE OR REPLACE VARIABLE @signature LONG BINARY;
   SELECT SECURE_SIGN_MESSAGE( 'This is exactly what I said', @private_key )
   INTO @signature;
   ```

3. The message sender sends the document, along with the signature and public key, to the message recipients.
4. The message recipients execute the following statements to verify that the message is authentic and that it has not been modified since being sent:

   ```
   CREATE OR REPLACE VARIABLE @verified INTEGER;
   SELECT SECURE_VERIFY_MESSAGE( 'This is exactly what I said', @signature,
   @public_key ) INTO @verified;
   ```

   If the @verified variable equals 1, then the message is authentic.

Related Information

Signing and Verifying Messages
SECURE_SIGN_MESSAGE Function [String] [page 545]
sp_generate_key_pair System Procedure [page 1777]

1.3.2.194 SET_BIT Function [Bit Array]

Sets the value of a specific bit in a bit array.

Syntax

```
SET_BIT( bit-expression [ , bit-position [ , value ] ] )
```

Parameters

bit-expression

The bit array in which to change the bit.

bit-position

The position of the bit to be set. This must be an unsigned integer.

value

The value to which the bit is to be set.

Returns

LONG VARBIT

Remarks

The default value of `bit-expression` is a bit array of length `bit-position`, containing all bits set to 0 (FALSE).

The default value of `value` is 1 (TRUE).

The result is NULL if any parameter is NULL.

The positions in the array are counted from the left side, starting at 1.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 00100011:

```
SELECT SET_BIT( '00110011', 4 , 0);
```

The following statement returns the value 00111011:

```
SELECT SET_BIT( '00110011', 5 , 1);
```

The following statement returns the value 00111011:

```
SELECT SET_BIT( '00110011', 5 );
```

The following statement returns the value 00001:

```
SELECT SET_BIT( 5 );
```

Related Information

GET_BIT Function [Bit Array] [page 389]
SET_BITS Function [Aggregate] [page 549]
INTEGER Data Type [page 150]
Bitwise Operators [page 33]
sa_get_bits System Procedure [page 1596]

1.3.2.195 SET_BITS Function [Aggregate]

Creates a bit array where specific bits, corresponding to values from a set of rows, are set to 1 (TRUE).

Syntax

```
SET_BITS( expression )
```

Parameters

expression

The expression used to determine which bits to set to 1. This is typically a column name.

Returns

LONG VARBIT

Remarks

Rows where the specified values are NULL are ignored.

If there are no rows, NULL is returned.

The length of the result is the largest position that was set to 1.

The SET_BITS function is equivalent to, but faster than, the following statement:

```
SELECT BIT_OR( SET_BIT( expression ) )
FROM table;
```

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements return a bit array with the 2nd, 5th, and 10th bits set to 1 (or 0100100001):

```
CREATE TABLE t( r INTEGER );
INSERT INTO t values( 2 );
INSERT INTO t values( 5 );
INSERT INTO t values(10 );
SELECT SET_BITS( r ) FROM t;
```

Related Information

Bitwise Operators [page 33]
GET_BIT Function [Bit Array] [page 389]
SET_BIT Function [Bit Array] [page 548]
sa_get_bits System Procedure [page 1596]

1.3.2.196 SIGN Function [Numeric]

Returns the sign (positive or negative) of the given number.

Syntax

```
SIGN( numeric-expression )
```

Parameters

numeric-expression

The number for which the sign is to be returned. `numeric-expression` may be of type INTEGER, DOUBLE, or NUMERIC.

Returns

SMALLINT

Remarks

For negative numbers, the SIGN function returns -1.

For zero, the SIGN function returns 0.

For positive numbers, the SIGN function returns 1.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value -1:

```
SELECT SIGN( -550 );
```

1.3.2.197 SIMILAR Function [String]

Returns a number indicating the similarity between two strings.

Syntax

```
SIMILAR( string-expression-1, string-expression-2 )
```

Parameters

string-expression-1

The first string to be compared.

string-expression-2

The second string to be compared.

Returns

SMALLINT

Remarks

The function returns an integer between 0 and 100 representing the similarity between the two strings. The result can be interpreted as the percentage of characters matched between the two strings. A value of 100 indicates that the two strings are identical.

This function can be used to correct a list of names (such as customers). Some customers may have been added to the list more than once with slightly different names. You can use the SIMILAR function to find similar customer names by joining the customer table to itself, producing a report of all similarities greater than 90 percent, but less than 100 percent.

The calculation performed for the SIMILAR function is more complex than just the number of characters that match.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 75, indicating that the two values are 75% similar:

```
SELECT SIMILAR( 'toast', 'coast' );
```

Related Information

String Functions [page 222]

1.3.2.198 SIN Function [Numeric]

Returns the sine of a number.

Syntax

```
SIN( numeric-expression )
```

Parameters

numeric-expression

The angle, in radians.

Returns

DOUBLE

Remarks

The SIN function returns the sine of the argument, where the argument is an angle expressed in radians. The SIN and ASIN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the SIN value of 0.52:

```
SELECT SIN( 0.52 );
```

Related Information

ASIN Function [Numeric] [page 251]
COS Function [Numeric] [page 298]
COT Function [Numeric] [page 299]
TAN Function [Numeric] [page 587]

1.3.2.199 SOAP_HEADER Function [SOAP]

Returns a SOAP header entry, or an attribute value for a header entry of the SOAP request.

Syntax

```
SOAP_HEADER( header-key [ , index [ , header-attribute ] ] )
```

Parameters

header-key

This VARCHAR parameter specifies the XML local name of the top level XML element for a given SOAP header entry.

index

This optional INTEGER parameter differentiates between SOAP header fields that have identical names. This can occur when multiple header entries have top level XML elements with the same localname. Usually, such elements have unique namespaces.

header-attribute

This optional VARCHAR parameter can specify any attribute node within a header entry element, including:

@namespace

A special SQL Anywhere attribute used to access the namespace of the given header entry.

mustUnderstand

A SOAP 1.1 header entry attribute indicating whether a header entry is mandatory or optional for the recipient to process.

encodingStyle

A SOAP 1.1 header entry attribute indicating the encoding style. This attribute can be accessed, but it is not used internally by SQL Anywhere.

actor

A SOAP 1.1 header entry attribute indicating the intended recipient of a header entry by specifying the recipient's URL.

Returns

LONG VARCHAR

Remarks

This function can be used with a single parameter `header-key` to return a header entry. A header entry is an XML string representation of an element, and all its sub-elements, contained within a SOAP header.

This function can also be used to extract a header entry attribute by specifying the optional `index` and `header-attribute` parameters.

This function returns the value of the named SOAP header field, or NULL if not called from a SOAP service. It is used when processing a SOAP request via a web service.

If a header for the given `header-key` does not exist, the return value is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

When used within a stored procedure that is called by an HTTP web service, the following example processes all the keys located in the SOAP request header. When it processes the Authentication key, it also obtains the key's value.

```
BEGIN
  DECLARE hd_key LONG VARCHAR;
  DECLARE hd_entry LONG VARCHAR;
header_loop:
  LOOP
    SET hd_key = NEXT_SOAP_HEADER( hd_key );
    IF hd_key IS NULL THEN
      -- no more header entries
      LEAVE header_loop;
    END IF;
    IF hd_key = 'Authentication' THEN
      SET hd_entry = SOAP_HEADER( hd_key );
    END IF;
  END LOOP header_loop;
END;
```

Related Information

Web Services Functions [page 221]
Tutorial: Using a Database Server to Access a SOAP/DISH Service
NEXT_SOAP_HEADER Function [SOAP] [page 477]
sa_set_soap_header System Procedure [page 1718]
Web Services System Procedures [page 1512]

1.3.2.200 SORTKEY Function [String]

Generates sort key values. That is, values that can be used to sort character strings based on alternate collation rules.

Syntax

```
SORTKEY( string-expression
[, { collation-id
   | collation-name [ ( collation-tailoring-string ) ] } ]
```

```
)
```

Parameters

string-expression

The string expression must contain characters that are encoded in the database's character set.

If `string-expression` is an empty string, the SORTKEY function returns a zero-length binary value. If `string-expression` is NULL, the SORTKEY function returns a NULL value. An empty string has a different sort order value than a NULL string from a database column.

The maximum length of the string that the SORTKEY function can handle is 254 bytes. Any longer part is ignored.

collation-name

A string or a character variable that specifies the name of the sort order to use. You can also specify the alias char_collation, or, equivalently, db_collation, to generate sortkeys as used by the CHAR collation in use by the database. Similarly, you can specify the alias nchar_collation to generate sortkeys as used by the NCHAR collation in use by the database.

collation-id

A variable, integer constant, or string that specifies the ID number of the sort order to use. If you do not specify `collation-id`, the default is Default Unicode multilingual.

collation-tailoring-string

Optionally, you can specify collation tailoring options (`collation-tailoring-string`) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs assembled in parentheses, following the collation name. For example, `'UCA(locale=es;case=LowerFirst;accent=respect)'`. The syntax for specifying these options is identical to the syntax defined for the COLLATION clause of the CREATE DATABASE statement.

> **i Note**
>
> All the collation tailoring options are supported when specifying the UCA collation. For all other collations, only case sensitivity tailoring is supported.

Returns

BINARY

Remarks

The SORTKEY function generates values that can be used to order results based on predefined sort order behavior. This allows you to work with character sort order behaviors that may not be available from the

database collation. The returned value is a binary value that contains coded sort order information for the input string that is retained from the SORTKEY function. For example, you can store the values returned by the SORTKEY function in a column with the source character string. When you want to retrieve the character data in the desired order, the SELECT statement only needs to include an ORDER BY clause on the columns that contain the results of running the SORTKEY function.

The SORTKEY function guarantees that the values it returns for a given set of sort order criteria work for the binary comparisons that are performed on VARBINARY data types.

Generating sortkeys for queries can be expensive. As an alternative for frequently requested sortkeys, consider creating a computed column to hold the sortkey values, and then referencing that column in the ORDER BY clause of the query.

The input of the SORTKEY function can generate up to six bytes of sort order information for each input character. The output of the SORTKEY function is of type VARBINARY and has a maximum length of 1024 bytes.

When specifying UCA for the collation during sort key generation, by default, collation tailorings are accent and case sensitive. For example, when UCA is specified by itself, the default tailoring applied is equivalent to `'UCA(case=UpperFirst;accent=Respect;punct=Primary)'`.

If a different tailoring is provided in the second parameter to SORTKEY, those settings override the default settings. For example, the following two statements are equivalent:

```
SELECT SORTKEY( 'abc', 'UCA(accent=Ignore)' );
SELECT SORTKEY( 'abc', 'UCA(case=UpperFirst;accent=Ignore;punct=Primary)' );
```

When specifying a non-UCA collation, by default, collation tailorings are also accent and case sensitive. However, for non-UCA collations, only the case sensitivity can be overridden using a collation tailoring. For example:

```
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

If the database was created without specifying tailoring options (for example, `dbinit -c -zn uca -dba DBA,passwd mydb.db`), the following two clauses may generate different sort orders, even if the database collation name is specified for the SORTKEY function:

```
ORDER BY string-expression
ORDER BY SORTKEY( string-expression, database-collation-name )
```

This is because the default tailoring settings used for database creation and for the SORTKEY function are different. To get the same behavior from SORTKEY as for the database collation, either provide a tailoring syntax for `collation-tailoring-string` that matches the settings for the database collation, or specify db_collation for `collation-name`. For example:

```
SORTKEY( expression, 'db_collation' )
```

> **i Note**
>
> Sort key values are generated differently depending on the version of SQL Anywhere. This can cause sorting issues if sort key values created by one version of SQL Anywhere are used in a database created by a different version of SQL Anywhere. You should regenerate sort key values if sorting issues occur.
>
> You should also regenerate sort key values when upgrading your database using unload/reload.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements queries the Employees table and returns the FirstName and Surname of all employees, sorted by the sortkey values for the Surname column using the dict collation (Latin-1, English, French, German dictionary):

```
SELECT Surname, GivenName FROM GROUPO.Employees ORDER BY SORTKEY( Surname,
'dict' );
```

The following example returns the sortkey value for abc, using the UCA collation and tailoring options:

```
SELECT SORTKEY( 'abc', 'UCA(locale=es;case=LowerFirst;accent=respect)' );
```

Related Information

International Languages and Character Sets
String Functions [page 222]
sort_collation Option
Alternate Collations
Collation Tailoring Options
Recommended Character Sets and Collations
COMPARE Function [String] [page 284]
CREATE DATABASE Statement [page 821]

1.3.2.201 SOUNDEX Function [String]

Returns a number representing the sound of a string.

Syntax

```
SOUNDEX( string-expression )
```

Parameters

string-expression

The string to be evaluated.

Returns

SMALLINT

Remarks

The SOUNDEX function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Vowels in `string-expression` are ignored unless they are the first letter of the string. Doubled letters are counted as one letter. For example, the word "apples" is based on the letters A, P, L, and S.

Multibyte characters are ignored by the SOUNDEX function.

Although it is not perfect, the SOUNDEX function normally returns the same number for words that sound similar and that start with the same letter.

The SOUNDEX function works best with English words. It is less useful for other languages.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns two identical numbers, 3827, representing the sound of each name:

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

Related Information

String Functions [page 222]

1.3.2.202 SPACE Function [String]

Returns a specified number of spaces.

Syntax

```
SPACE( integer-expression )
```

Parameters

integer-expression

The number of spaces to return.

Returns

LONG VARCHAR

Remarks

If `integer-expression` is negative, a null string is returned.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns a string containing 10 spaces:

```
SELECT SPACE( 10 );
```

Related Information

String Functions [page 222]

1.3.2.203 SQLDIALECT Function [Miscellaneous]

Returns either Watcom SQL or Transact-SQL, to indicate the SQL dialect of a statement.

Syntax

```
SQLDIALECT( sql-statement-string )
```

Parameters

sql-statement-string

The SQL statement that the function uses to determine its dialect.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the string Transact-SQL:

```
SELECT
   SQLDIALECT( 'SELECT EmployeeName = Surname FROM GROUPO.Employees' )
FROM SYS.DUMMY;
```

Related Information

TRANSACTSQL Function [Miscellaneous] [page 599]
WATCOMSQL Function [Miscellaneous] [page 622]

1.3.2.204 SQLFLAGGER Function [Miscellaneous]

Returns the conformity of a given SQL statement to a specified standard such as the ANSI/ISO SQL Standard.

Syntax

```
SQLFLAGGER( sql-standard-string, sql-statement-string )
```

Parameters

sql-standard-string

The standard level against which to test compliance. Possible values are the same as for the sql_flagger_error_level database option:

SQL:2008/Core

Test for conformance to core SQL/2008 syntax.

SQL:2008/Package

Test for conformance to full SQL/2008 syntax.

SQL:2003/Core

Test for conformance to core SQL/2003 syntax.

SQL:2003/Package

Test for conformance to full SQL/2003 syntax.

SQL:1999/Core

Test for conformance to core SQL/1999 syntax.

SQL:1999/Package

Test for conformance to full SQL/1999 syntax.

SQL:1992/Entry

Test for conformance to entry-level SQL/1992 syntax.

SQL:1992/Intermediate

Test for conformance to intermediate-level SQL/1992 syntax.

SQL:1992/Full

Test for conformance to full-SQL/1992 syntax.

UltraLite

Test for conformance to UltraLite.

sql-statement-string

The SQL statement to check for conformance.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement shows an example of the message that is returned when a disallowed extension is found:

```
SELECT SQLFLAGGER(
   'SQL:2003/Package', 'SELECT top 1 dummy_col FROM sys.dummy ORDER BY
dummy_col' );
```

This statement returns the message `'0AW03 Disallowed language extension detected in syntax near 'top' on line 1'`.

The following statement returns `'00000'` because it does not contain any disallowed extensions:

```
SELECT SQLFLAGGER( 'SQL:2003/Package', 'SELECT dummy_col FROM sys.dummy' );
```

Related Information

SQL Compliance Testing Using the SQL Flagger
sql_flagger_error_level Option
The Embedded SQL Preprocessor
sa_ansi_standard_packages System Procedure [page 1529]

1.3.2.205 SQRT Function [Numeric]

Returns the square root of a number.

Syntax

```
SQRT( numeric-expression )
```

Parameters

numeric-expression

The number for which the square root is to be calculated.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

Standards

ANSI/ISO SQL Standard

The SQRT function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 3:

```
SELECT SQRT( 9 );
```

1.3.2.206 STACK_TRACE Function [Miscellaneous]

Returns information about the stack trace for the current statement.

Syntax

```
STACK_TRACE(
[ stack-frames
[, detail-level
[, connection-id ] ] ]
)
```

Parameters

stack-frames

Controls whether to include procedures, outer-statements, or both.

'procedure'

Return procedures but not the outer-most statement. This is the default behavior.

'caller'

Return only the outer-most statement (the statement that arrived from the client).

'procedure+caller', 'caller+procedure'

Return all statements.

detail-level

Controls the level of detail to include in the returned data.

'stack'

Include procedure names and line numbers. This is the default behavior.

'stack+sql', 'sql+stack'

Include the procedure names and line numbers, as well as the SQL text of the statement being executed at each level.

connection-id

Use the connection_id option to filter the results returned to the specified connection ID.

Returns

LONG VARCHAR representing the stack trace of the current statement.

Remarks

The result contains lines of text delimited by line feed (\n) characters. Each line of the returned value contains the qualified procedure name or batch type, followed by the line number of the statement. The last line of the returned value is not terminated by a line feed character. The first line of the stack trace represents the line where the function was invoked. If a compound statement is not part of a procedure, function, trigger, or event, then the type of batch (watcom_batch or tsql_batch) is returned instead of the procedure name.

This function returns line numbers as found in the proc_defn column of the SYSPROCEDURE system table for the procedure. These line numbers might differ from those of the source definition used to create the procedure.

This function returns the same information as the sa_stack_trace system procedure.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example illustrates a procedure call stack trace:

```
CREATE OR REPLACE PROCEDURE proc3()
BEGIN
   DECLARE v INTEGER;
   SET v = 1;
   SELECT * FROM sa_split_list( STACK_TRACE('caller+procedure', 'stack+sql'),
'\n' );
END;
CREATE OR REPLACE PROCEDURE proc2()
BEGIN
    CALL proc3();
END;
CREATE OR REPLACE PROCEDURE proc1()
BEGIN
    CALL proc2();
END;
CALL proc1();
```

Results:

```
line_num row_value
-------------------------------------------------------------------------------
-----------------------------------------------
          1   "DBA"."proc3" : 5 : select
sa_split_list.line_num,sa_split_list.row_value from
sa_split_list(STACK_TRACE('caller+procedure','stack
+sql'),'\x0A')

          2   "DBA"."proc2" : 3 : call
proc3()
```

```
          3   "DBA"."proc1" : 3 : call
proc2()

          4    call proc1(
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
SYSPROCEDURE System View [page 1931]

1.3.2.207 STDDEV Function [Aggregate]

An alias for STDDEV_SAMP.

Related Information

STDDEV_SAMP Function [Aggregate] [page 570]

1.3.2.208 STDDEV_POP Function [Aggregate]

Computes the standard deviation of a population consisting of a numeric-expression, as a DOUBLE.

Syntax

Expression

```
STDDEV_POP( numeric-expression )
```

Window function

```
STDDEV_POP( numeric-expression ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

numeric-expression

The expression whose population-based standard deviation is calculated over a set of rows. The expression is commonly a column name.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

The population-based standard deviation (s) is computed according to the following formula:

$s = [(1/N) * SUM(x_I - mean(x))^2]^{½}$

This standard deviation does not include rows where `numeric-expression` is NULL. It returns NULL for a group containing no rows.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

The STDDEV_POP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions". When used as window function, STDDEV_POP comprises part of optional ANSI/ISO SQL Foundation Feature T611, "Elementary OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports

ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software also does not permit the arguments of an aggregate function to include both a column reference from the query block containing the STDDEV_POP function, combined with an outer reference.

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    STDDEV_POP( quantity ) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.2794...
2000	2	27.050847	15.0270...
...	...	...	...

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
Aggregate Functions [page 208]
WINDOW Clause [page 1489]

1.3.2.209 STDDEV_SAMP Function [Aggregate]

Computes the standard deviation of a sample consisting of a numeric-expression, as a DOUBLE.

Syntax

Expression

```
STDDEV_SAMP( numeric-expression )
```

Window function

```
STDDEV_SAMP( numeric-expression ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

numeric-expression

The expression whose sample-based standard deviation is calculated over a set of rows. The expression is commonly a column name.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

The standard deviation (s) is computed according to the following formula, which assumes a normal distribution:

$s = [(1/(N - 1)) * SUM(x_I - mean(x))^2]^{½}$

This standard deviation does not include rows where `numeric-expression` is NULL. It returns NULL for a group containing either 0 or 1 rows.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

The STDDEV_SAMP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions". When used as window function, STDDEV_SAMP comprises part of optional SQL foundation feature T611, "Elementary OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Standard language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software also does not permit the arguments of an aggregate function to include both a column reference from the query block containing the STDDEV_SAMP function, combined with an outer reference.

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    STDDEV_SAMP( quantity ) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.3218...
2000	2	27.050847	15.0696...
...	...	...	...

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
Aggregate Functions [page 208]
WINDOW Clause [page 1489]
AVG Function [Aggregate] [page 254]

1.3.2.210 STR Function [String]

Returns the string equivalent of a number.

Syntax

```
STR( numeric-expression [, length [, decimal ] ] )
```

Parameters

numeric-expression

Any approximate numeric (float, real, or double precision) expression between -1E126 and 1E127.

length

The number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10.

decimal

The number of decimal digits to be returned. The default is 0.

Returns

VARCHAR

Remarks

If the integer portion of the number cannot fit in the length specified, then the result is a string of the specified length containing all asterisks. For example, the following statement returns ***.

```
SELECT STR( 1234.56, 3 );
```

Note

The maximum length that is supported is 128. Any length that is not between 1 and 128 yields a result of NULL.

Standards

ANSI/ISO SQL Standard

A feature of the standard.

Example

The following statement returns a string of six spaces followed by 1235, for a total of ten characters:

```
SELECT STR( 1234.56 );
```

The following statement returns the result 1234.6:

```
SELECT STR( 1234.56, 6, 1 );
```

Related Information

String Functions [page 222]

1.3.2.211 STRING Function [String]

Concatenates one or more strings into one large string.

Syntax

```
STRING( string-expression [, ... ] )
```

Parameters

string-expression

The string to be evaluated.

If only one argument is supplied, it is converted into a single expression. If multiple arguments are supplied, they are concatenated into a single string.

Returns

LONG VARCHAR, LONG NVARCHAR, or LONG BINARY, depending on the data type of the input expression.

Remarks

Numeric or date parameters are converted to strings before concatenation. The STRING function can also be used to convert any single expression to a string by supplying that expression as the only parameter.

If all parameters are NULL, STRING returns NULL. If any parameters are non-NULL, then any NULL parameters are treated as empty strings.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value `testing123`:

```
SELECT STRING( 'testing', NULL, 123 );
```

Related Information

String Functions [page 222]

1.3.2.212 STRTOUUID Function [String]

Converts a string value to a unique identifier (UUID or GUID) value.

Syntax

```
STRTOUUID( string-expression )
```

Parameters

string-expression

A string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Returns

UNIQUEIDENTIFIER

Remarks

Converts a string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, where `x` is a hexadecimal digit, to a unique identifier value.

This function is useful for inserting UUID values into a database.

If the string is not a valid UUID string, a conversion error is returned unless the conversion_error option is set to OFF, in which case it returns NULL. This function supports NCHAR inputs and/or outputs. Curly braces can be used as the first and last characters in the `string-expression`. In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type was defined as a user-defined data type and the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values. In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type and the database server carries out conversions as needed. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

UltraLite: In databases created before version 9.0.2, the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values. In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements are equivalent and return the result 0x6c2b64a93c6f47dc901536b9ed49fec2:

```
SELECT STRTOUUID( '6c2b64a9-3c6f-47dc-9015-36b9ed49fec2' );
SELECT STRTOUUID( '{6c2b64a9-3c6f-47dc-9015-36b9ed49fec2}' );
```

Related Information

String Functions [page 222]
UNIQUEIDENTIFIER Data Type [page 183]
UUIDTOSTR Function [String] [page 615]
NEWID Function [Miscellaneous] [page 467]

1.3.2.213 STUFF Function [String]

Deletes multiple characters from one string and replaces them with another string.

Syntax

```
STUFF( string-expression-1, start, length, string-expression-2 )
```

Parameters

string-expression-1

The string to be modified by the STUFF function.

start

The character position at which to begin deleting characters. The first character in the string is position 1.

length

The number of characters to delete.

string-expression-2

The string to be inserted. To delete a portion of a string using the STUFF function, use a replacement string of NULL.

Returns

LONG BINARY, LONG VARCHAR, or LONG NVARCHAR, depending on the data type of the input expressions.

Remarks

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value chocolate pie:

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

Related Information

String Functions [page 222]
INSERTSTR Function [String] [page 423]

1.3.2.214 SUBSTRING Function [String]

Returns a substring of a string.

Syntax

```
{ SUBSTRING | SUBSTR }( string-expression , start [ , length ] )
```

Parameters

string-expression

The string from which a substring is to be returned.

start

The start position of the substring to return, in characters.

length

The length of the substring to return, in characters. If `length` is specified, the substring is restricted to that length.

Returns

LONG BINARY
LONG VARCHAR
LONG NVARCHAR
UltraLite returns LONG BINARY and LONG VARCHAR

Remarks

To obtain characters at the end of a string, use the RIGHT function.

If `string-expression` is of binary data type, then the SUBSTRING function behaves as BYTE_SUBSTR.

This function supports NCHAR inputs and/or outputs. Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics. The behavior of this function depends on the setting of the ansi_substring database option. When the ansi_substring option is set to On (the default), the behavior of the SUBSTRING function corresponds to ANSI/ISO SQL Standard behavior. The behavior is as follows:

ansi_substring option setting	start value	length value
On	The first character in the string is at position 1. A negative or zero start offset is treated as if the string were padded on the left with non-characters.	A positive `length` specifies that the substring ends `length` characters to the right of the starting position. A negative `length` returns an error.
Off	The first character in the string is at position 1. A negative starting position specifies a number of characters from the end of the string instead of the beginning. If `start` is zero and length is non-negative, a start value of 1 is used. If start is zero and `length` is negative, a start value of -1 is used.	A positive `length` specifies that the substring ends `length` characters to the right of the starting position. A negative `length` returns at most `length` characters up to, and including, the starting position, from the left of the starting position.

UltraLite: Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics. In UltraLite, the database does not have an ansi_substring option, but the SUBSTR function behaves as if ansi_substring is set to on by default. The function's behavior corresponds to ANSI/ISO SQL Standard behavior:

Start value

The first character in the string is at position 1. A negative or zero start offset is treated as if the string were padded on the left with non-characters.

Length value

A positive `length` specifies that the substring ends `length` characters to the right of the starting position.

A negative `length` returns an error.

A `length` of zero returns an empty string.

Standards

ANSI/ISO SQL Standard

Core Feature. However, the ANSI/ISO SQL Standard implementation differs slightly from the software: in the Standard, SUBSTRING is defined with three parameters using the keywords FROM and FOR, neither of which are required by the software.

Example

The following table shows the values returned by the SUBSTRING function:

Example	Result
SUBSTRING('front yard', 1, 4)	fron
SUBSTRING('back yard', 6, 4)	yard
SUBSTR('abcdefgh', 0, -2)	Returns an error if the SQL Anywhere ansi_substring option is On
SUBSTR('abcdefgh', -2, 2)	Returns an empty string if the SQL Anywhere ansi_substring option is On

UltraLite: The following table shows the values returned by the SUBSTRING function:

Example	Result
SUBSTRING('front yard', 1, 4)	fron
SUBSTRING('back yard', 6, 4)	yard
SUBSTR('abcdefgh', 0, -2)	Returns an error
SUBSTR('abcdefgh', -2, 2)	Returns an empty string

Related Information

String Functions [page 222]
BYTE_SUBSTR Function [String] [page 272]
LEFT Function [String] [page 435]
RIGHT Function [String] [page 532]
CHARINDEX Function [String] [page 282]

1.3.2.215 SUM Function [Aggregate]

Returns the total of the specified expression for each group of rows.

Syntax

Expression

```
SUM( [ ALL | DISTINCT ] expression )
```

Window function

```
SUM( [ ALL ] expression ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

UltraLite syntax: Expression

```
SUM( [ DISTINCT ] expression )
```

Parameters

expression

The name of the expression to be summed. This is commonly a column name.

[ALL] expression

The name of the expression to be summed. This is commonly a column name.

DISTINCT expression

Computes the sum of the unique values of `expression` within each group.

Returns

- INTEGER
- DOUBLE
- NUMERIC

Remarks

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

This function can generate an overflow error, resulting in an error being returned. You can use the CAST function on `numeric-expression` to avoid the overflow error.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

Core Feature. When used as a window function, SUM comprises part of optional ANSI/ISO SQL Language Feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the SUM function, combined with an outer reference.

Example

The following statement returns the value 3749146.740:

```
SELECT SUM( Salary )
FROM GROUPO.Employees;
```

Related Information

Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]
COUNT Function [Aggregate] [page 300]
AVG Function [Aggregate] [page 254]
Troubleshooting Database Upgrades: Aggregate Functions and Outer References

1.3.2.216 SUSER_ID Function [System]

Returns the numeric user ID for the specified user name.

Syntax

```
SUSER_ID( [ user-name ] )
```

Parameters

user-name

The user name for the user ID you are searching for.

Returns

UNSIGNED INTEGER

Remarks

If you do not specify `user-name`, the ID of the current user is returned.

This function is provided for compatibility with other vendors. You can also use the USER_ID function, which does exactly the same thing.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns 101, the ID for the GROUPO user:

```
SELECT SUSER_ID( 'GROUPO' );
```

Related Information

SUSER_NAME Function [System] [page 584]
USER_ID Function [System] [page 612]

1.3.2.217 SUSER_NAME Function [System]

Returns the user name for the specified user ID.

Syntax

```
SUSER_NAME( [ user-id ] )
```

Parameters

user-id

The user ID of the user you are searching for.

Returns

VARCHAR

Remarks

If you do not specify `user-id`, the user name of the current user is returned.

This function is provided for compatibility with other vendors. You can also use the USER_NAME function, which does exactly the same thing.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns GROUPO, the user name for a user with ID 101:

```
SELECT SUSER_NAME( 101 );
```

Related Information

SUSER_ID Function [System] [page 583]
USER_NAME Function [System] [page 613]

1.3.2.218 SWITCHOFFSET Function [Date and Time]

Returns a TIMESTAMP WITH TIME ZONE value that is converted from its original time zone offset to the specified time zone offset.

Syntax

```
SWITCHOFFSET( tmz-expression, time-zone-offset )
```

Parameters

tmz-expression

The TIMESTAMP WITH TIME ZONE value to be converted.

time-zone-offset

The time zone offset of the result. The value can be an integer representing the minutes before or after Coordinated Universal Time (UTC), a string in the form { + | - } hh:nn, or Z for the Zulu Time Zone. The Zulu Time Zone is the same time zone as UTC.

Returns

TIMESTAMP WITH TIME ZONE

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example changes a time zone offset value from -04:00 hours to -07:00 hours. The value returned is 2009-04-03 11:45:12.123-07:00:

```
SELECT CAST ( '2009-04-03 14:45:12.123-04:00' AS datetimeoffset ) AS EDT,
SWITCHOFFSET( EDT,'-07:00' ) AS PDT;
```

Related Information

TIMESTAMP WITH TIME ZONE Data Type [page 177]
SYSDATETIMEOFFSET Function [Date and Time] [page 586]

1.3.2.219 SYSDATETIMEOFFSET Function [Date and Time]

Returns the current date, time, and time zone offset of the database server using the system clock.

Syntax

SYSDATETIMEOFFSET ( )

Returns

TIMESTAMP WITH TIME ZONE

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the current date and time and the time zone offset of the database server:

```
SELECT SYSDATETIMEOFFSET ( );
```

The following example converts the SYSDATETIMEOFFSET value to the time zone of the client computer:

```
SELECT SWITCHOFFSET ( SYSDATETIMEOFFSET ( ),
CAST( connection_property ('TimeZoneAdjustment') AS INT ));
```

Related Information

TIMESTAMP WITH TIME ZONE Data Type [page 177]
SWITCHOFFSET Function [Date and Time] [page 585]

1.3.2.220 TAN Function [Numeric]

Returns the tangent of a number.

Syntax

```
TAN( numeric-expression )
```

Parameters

numeric-expression

An angle, in radians.

Returns

DOUBLE

Remarks

The ATAN and TAN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value of the tan of 0.52:

```
SELECT TAN( 0.52 );
```

Related Information

COS Function [Numeric] [page 298]
SIN Function [Numeric] [page 553]

1.3.2.221 TEXTPTR Function [Text and Image]

Returns a 16-byte binary pointer to the specified column. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

Syntax

```
TEXTPTR( column-name )
```

Parameters

column-name

The name of a column containing CHAR, NCHAR, or BINARY data.

Returns

BINARY

Remarks

This function is included for Transact-SQL compatibility.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following Embedded SQL example uses TEXTPTR to locate the Description column associated with ProductID 500 in the MarketingInformation table:

```
EXEC SQL BEGIN DECLARE SECTION;
char            hostvar[100];
EXEC SQL END DECLARE SECTION;
EXEC SQL create variable txtptr binary(16);
EXEC SQL set txtptr =
    ( SELECT textptr(Description)
        FROM GROUPO.MarketingInformation
        WHERE ProductID = '500' );
EXEC SQL PREPARE S1 FROM
    'READTEXT GROUPO.MarketingInformation.Description txtptr 181 55';
EXEC SQL EXECUTE S1 INTO :hostvar;
printf( "hostvar: %s\n", hostvar );
```

The text pointer is stored in the variable txtptr and supplied as a parameter to the READTEXT statement which returns 55 bytes, starting at column offset 181. READTEXT returns the following string:

```
Lightweight 100% organically grown cotton construction.
```

1.3.2.222 TO_CHAR Function [String]

Converts character data from any supported character set into the CHAR character set for the database.

Syntax

```
TO_CHAR( string-expression [ , source-charset-name ] )
```

Parameters

string-expression

The string to be converted.

source-charset-name

The character set of the string.

Returns

LONG VARCHAR

Remarks

If `source-charset-name` is specified, then this function is equivalent to:

```
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
 'db_charset', source-charset-name )
   AS CHAR );
```

If `source-charset-name` is not specified, then this function is equivalent to:

```
CAST( string-expression AS CHAR );
```

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

If you have a BINARY value containing data in the cp850 character set, the following statement converts the data to the CHAR character set and data type:

```
SELECT TO_CHAR( 'cp850_data', 'cp850' );
```

Related Information

Recommended Character Sets and Collations
CONNECTION_EXTENDED_PROPERTY Function [String] [page 290]
CSCONVERT Function [String] [page 309]
NCHAR Function [String] [page 466]
TO_NCHAR Function [String] [page 591]
UNICODE Function [String] [page 608]
UNISTR Function [String] [page 609]

1.3.2.223 TO_NCHAR Function [String]

Converts character data from any supported character set into the NCHAR character set.

Syntax

```
TO_NCHAR( string-expression [ , source-charset-name ] )
```

Parameters

string-expression

The string to be converted.

source-charset-name

The character set of the string.

Returns

LONG NVARCHAR

Remarks

If `source-charset-name` is specified then this function is equivalent to:

```
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
 'nchar_charset', source-charset-name )
 AS NCHAR );
```

If `source-charset-name` is not provided then this function is equivalent to:

```
CAST( string-expression AS NCHAR );
```

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

If you have a BINARY value containing data in the cp850 character set, the following example to converts the data to the NCHAR character set and data type:

```
SELECT TO_NCHAR( 'cp850_data', 'cp850' );
```

Related Information

Recommended Character Sets and Collations
CONNECTION_EXTENDED_PROPERTY Function [String] [page 290]
CSCONVERT Function [String] [page 309]
NCHAR Function [String] [page 466]
TO_CHAR Function [String] [page 589]
UNICODE Function [String] [page 608]
UNISTR Function [String] [page 609]

1.3.2.224 TODATETIMEOFFSET Function [Date and Time]

Converts a TIMESTAMP value to a TIME STAMP WITH TIME ZONE value using the specified time zone offset.

Syntax

```
TODATETIMEOFFSET( timestamp-expression, time-zone-offset )
```

Parameters

timestamp-expression

The TIMESTAMP expression to be converted.

time-zone-offset

The time zone offset. The value can be an INTEGER representing minutes before or after UTC, a VARCHAR string in the form of { + | - }hh:nn, or the string "Z" for the Zulu Time Zone. The Zulu Time Zone is the same time zone as UTC.

Returns

TIMESTAMP WITH TIME ZONE

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example converts a TIMESTAMP value to a TIMESTAMP WITH TIME ZONE value:

```
SELECT CAST('2009-04-03 14:45:12.123' AS TIMESTAMP) AS orig,
        TODATETIMEOFFSET (orig,'+11:00');
```

Related Information

TIMESTAMP WITH TIME ZONE Data Type [page 177]

1.3.2.225 TODAY Function [Date and Time]

Returns the current date as a DATE value.

Syntax

```
TODAY( [ * ] )
```

Returns

DATE

Remarks

TODAY(*) and TODAY() are semantically equivalent. TODAY is equivalent to the CURRENT DATE special value.

Each instance of the TODAY function in a request is evaluated at most once. Multiple instances of TODAY in the same request may or may not share the identical DATE value.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements return the current day according to the system clock:

```
SELECT TODAY( * );
SELECT CURRENT DATE;
```

1.3.2.226 TOLOCALTIME Function [Date and time]

Converts a TIMESTAMP WITH TIME ZONE value, or a TIMESTAMP value (which is assumed to be in Coordinated Universal Time (UTC)), to a timestamp value that corresponds to the server's local time using the standard time or daylight saving time rules for the server's locale.

Syntax

```
TOLOCALTIME( timestamp-expression )
```

Parameters

timestamp-expression

The TIMESTAMP WITH TIME ZONE or TIMESTAMP expression to be converted. The TIMESTAMP WITH TIME ZONE value is converted to UTC before determining the local time. A TIMESTAMP value is assumed to be in UTC.

Returns

TIMESTAMP

Remarks

The TOLOCALTIME function accepts any supported TIMESTAMP or TIMESTAMP WITH TIME ZONE value. However, it's intended to be used with date/time values that are recent since rules for daylight savings time have changed over the years for most locales that implemented daylight savings at one time or another.

Historical local time values are platform-dependent. On Windows, the current standard time/daylight savings time rules are used for historical dates. On Linux and UNIX platforms, the standard time/daylight savings time rules in effect on the specified date can be used for historical dates. For example, for database servers running on Linux in North America, dates before 2007 may be converted using the standard time/daylight time rules in effect before the most recent change to the rules in 2007. Also, dates before 1970 may or may not use the correct daylight saving rules.

Similarly, future dates can be converted using current rules even if those rules don't apply when the date arrives. Dates that follow the current date can be subject to change. Some locales never implemented daylight savings time (for example, China/Beijing Time UTC+8) so conversion from UTC to local time is straightforward.

Note that a string literal value specified as an argument to the TOLOCALTIME function must be cast to a TIMESTAMP WITH TIME ZONE if that is what the string value represents. If it is not cast to a TIMESTAMP WITH TIME ZONE, it is converted to a TIMESTAMP as if the zone offset isn't present.

Examples

Example 1

The following example assumes that the database server is running in the North American Eastern time zone and shows how daylight rules affect the outcome. The input values represent dates and times that are 1 hour ahead of UTC. The date and time are converted to UTC by subtracting 1 hour and then converted to Eastern time, using the Standard Time/Daylight Time rules for the North American Eastern time zone.

The first date occurs when Eastern Standard time is in effect (5 hours behind UTC). The second and third dates occur when Eastern Daylight time is in effect (4 hours behind UTC).

```
SELECT CAST('2019/02/09 21:26:45.6789+1:00' AS TIMESTAMP WITH TIME ZONE) AS CT,
TOLOCALTIME( CT )
UNION ALL
SELECT CAST('2019/03/10 21:26:45.6789+1:00' AS TIMESTAMP WITH TIME ZONE) AS CT,
TOLOCALTIME( CT )
UNION ALL
```

```
SELECT CAST('2019/08/09 21:26:45.6789+1:00' AS TIMESTAMP WITH TIME ZONE) AS CT,
TOLOCALTIME( CT )
ORDER BY CT
CT,                                  TOLOCALTIME(CT)
'2019-02-09 21:26:45.678+01:00',     '2019-02-09 15:26:45.678'
'2019-03-10 21:26:45.678+01:00',     '2019-03-10 16:26:45.678'
'2019-08-09 21:26:45.678+01:00',     '2019-08-09 16:26:45.678'
```

Example 2

The following example assumes that the database server is running in the North American Eastern time zone and shows how daylight rules affect the outcome. The input values represent dates and times that are 5 or 4 hours behind UTC. The date and time are converted to UTC by adding the zone offset and then converted to Eastern time, using the Standard Time/Daylight Time rules for the North American Eastern time zone.

The first date occurs when Eastern Standard time is in effect (5 hours behind UTC). The second and third dates occur when Eastern Daylight time is in effect (4 hours behind UTC). As a result, each local time result has the same time of day as the input value.

```
SELECT CAST('2019/02/09 21:26:45.6789-5:00' AS TIMESTAMP WITH TIME ZONE) AS ET,
TOLOCALTIME( ET )
UNION ALL
SELECT CAST('2019/03/10 21:26:45.6789-4:00' AS TIMESTAMP WITH TIME ZONE) AS ET,
TOLOCALTIME( ET )
UNION ALL
SELECT CAST('2019/08/09 21:26:45.6789-4:00' AS TIMESTAMP WITH TIME ZONE) AS ET,
TOLOCALTIME( ET )
ORDER BY ET;
ET,                                  TOLOCALTIME(ET)
'2019-02-09 21:26:45.678-05:00',     '2019-02-09 21:26:45.678'
'2019-03-10 21:26:45.678-04:00',     '2019-03-10 21:26:45.678'
'2019-08-09 21:26:45.678-04:00',     '2019-08-09 21:26:45.678'
```

Example 3

The following example assumes that the database server is running in the North American Eastern time zone and shows how daylight rules affect the outcome. The input value, a TIMESTAMP, is assumed to be in UTC. Since Eastern Daylight time (4 hours behind UTC) is in effect for the specified date, the local time result is 4 hours earlier.

```
SELECT CAST('2019/08/09 21:26:45.6789' AS TIMESTAMP) AS UT, TOLOCALTIME( UT );
UT,                            TOLOCALTIME(UT)
'2019-08-09 21:26:45.678',     '2019-08-09 17:26:45.678'
```

1.3.2.227 TRACEBACK Function [Miscellaneous]

Returns statements on the stack of the most recent exception (error) that occurred during a stored procedure, trigger, or custom function execution.

Syntax

```
TRACEBACK( [ * ] )
```

Returns

LONG VARCHAR

Remarks

The returned call stack is annotated with the object names and line numbers. Statements from stored procedures with HIDDEN definitions are logged as `<hidden>` in the stack trace.

The statements in the TRACEBACK are generated from the database server representation of the statements in the stored procedures, which may not precisely match the text in the procedure definition.

TRACEBACK(*) and TRACEBACK() are semantically equivalent.

This function is useful for debugging procedures and triggers, particularly those that are written in the Transact-SQL dialect.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

To use the TRACEBACK function, execute the following statement after an error occurs while executing a procedure:

```
SELECT TRACEBACK( * );
```

Output similar to the following output is returned after using the TRACEBACK function:

```
"user1"."proc1" : 10 : set ret_val = in_val / (in_val - in_val)
"user2"."proc2" : 5 : <hidden>
"user3"."proc1" : 7 : call user2.proc2( 10 )
```

1.3.2.228 TRACED_PLAN Function [Miscellaneous] (Deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. This function is used by *SQL Central* to generate a graphical plan for a query using tracing data.

Syntax

```
TRACED_PLAN( logging_session_id, query_id )
```

Parameters

logging_session_id

Combined with `query_id`, this INTEGER parameter identifies a row from the sa_diagnostic_query table for which to generate the plan.

query_id

Combined with `logging_session_id`, this INTEGER parameter identifies a row from the sa_diagnostic_query table for which to generate the plan.

Returns

LONG VARCHAR

Remarks

This function is for use by *SQL Central*.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

sa_diagnostic_query Table (Deprecated) [page 1503]

1.3.2.229 TRANSACTSQL Function [Miscellaneous]

Rewrites a Watcom SQL statement in the Transact-SQL dialect.

Syntax

```
TRANSACTSQL( sql-statement-string )
```

Parameters

sql-statement-string

The SQL statement that is to be rewritten in Transact-SQL.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the string `'select GivenName,LastName=Surname from GROUPO.Employees'`:

```
SELECT TRANSACTSQL( 'SELECT GivenName, Surname as LastName FROM
GROUPO.Employees' ) FROM SYS.DUMMY;
```

Related Information

SQLDIALECT Function [Miscellaneous] [page 562]
WATCOMSQL Function [Miscellaneous] [page 622]

1.3.2.230 TREAT Function [Data Type Conversion]

Changes the declared type of a geometry expression to a subtype. This function is for use with spatial data.

Syntax

```
TREAT ( geometry-expression AS subtype )
```

Parameters

geometry-expression

The expression to be converted.

subtype

The target subtype to convert `geometry-expression` into.

Returns

Depends on the data type requested.

Remarks

The TREAT function can only be used on geometries.

If the dynamic type of the expression is not a subtype of the target data type, an error is returned. The CAST function can also be used to change the declared type of a geometry expression. However, the CAST function allows changes outside of the subtype hierarchy. For example, CAST can be used to convert a point to a multipoint. These types of conversions may change the dynamic type of an expression in unexpected ways, so TREAT is preferable when moving from a supertype to a subtype. The TREAT function also executes more efficiently than the CAST function.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Execute the following statement to create a table and load two values into it:

```
DROP TABLE IF EXISTS treatExample;
CREATE TABLE treatExample( pk INT PRIMARY KEY, geo ST_Geometry );
INSERT INTO treatExample VALUES(0, NEW ST_Point(3,4) );
INSERT INTO treatExample VALUES(1, NEW ST_MultiPoint( new ST_Point( 5, 6 ) ) );
```

The following query returns an error.

```
SELECT geo.ST_X() FROM treatExample T WHERE pk = 0;
```

The following query succeeds:

```
SELECT TREAT( geo AS ST_Point ).ST_X() FROM treatExample WHERE pk = 0;
```

The following query returns an error.

```
SELECT TREAT( geo AS ST_Point ).ST_X() FROM treatExample T WHERE pk = 0;
```

The following query succeeds because a CAST statement is used instead of a TREAT statement:

```
SELECT CAST( geo AS ST_Point ) FROM treatExample WHERE pk = 1;
```

Related Information

Using the TREAT Expression for Subtypes
CAST Function [Data Type Conversion] [page 275]
Type '%1' has no method named '%2' (near '%3')

1.3.2.231 TRIM Function [String]

Removes leading and trailing blanks or specified characters from a string.

Syntax

```
TRIM( string-expression [ , trim-char-set ] )
```

Parameters

string-expression

The string to be trimmed.

trim-char-set

The set of characters to trim.

Returns

VARCHAR, NVARCHAR, LONG VARCHAR, or LONG NVARCHAR, depending on the data type of the input expression.

Remarks

By default, `trim-char-set` is the space character. You can specify the set of characters to be trimmed.

This function supports NCHAR inputs and/or outputs.

UltraLite does not support NCHAR inputs and/or outputs and `trim-char-set`.

Standards

ANSI/ISO SQL Standard

Core Feature.

The software does not support the additional parameters `trim specification` and `trim character`, as defined in the ANSI/ISO SQL Standard. The implementation of TRIM provided in the software corresponds to a TRIM specification of BOTH.

For the other TRIM specifications defined by the ANSI/ISO SQL Standard (LEADING and TRAILING), the software provides the LTRIM and RTRIM functions respectively.

Example

The following statement returns the value chocolate with no leading or trailing blanks:

```
SELECT TRIM( '   chocolate   ' );
```

The following statement returns the value def after the specified leading and trailing characters are removed:

```
SELECT TRIM('abccbadefabcabccba', 'abc' );
```

Related Information

String Functions [page 222]
LTRIM Function [String] [page 448]
RTRIM Function [String] [page 540]

1.3.2.232 TRIM_ARRAY Function [Composite]

Returns an implicitly bounded array that consists of a specified number of elements in an array.

Syntax

```
TRIM_ARRAY( array-expression, integer-expression )
```

Parameters

array-expression

The array to trim.

integer-expression

The number of elements that the resulting array should contain. The resulting array contains the first `integer-expression` number of elements in the `array-expression`.

If the `integer-expression` is zero, the resulting array is empty. If the `integer-expression` is less than zero, or is greater than the cardinality of the array, then an error is generated.

Returns

ARRAY

Remarks

If either argument to TRIM_ARRAY is NULL, then the result is NULL.

Standards

ANSI/ISO SQL Standard

Feature S404.

Example

The following example illustrates how to use the TRIM_ARRAY function to create an array that consists of the first two elements in an array that contains four elements:

```
DECLARE UntrimmedArray ARRAY( 4 ) OF INT;
DECLARE TrimmedArray ARRAY( 2 ) OF INT;
SET UntrimmedArray = ARRAY( 1, 2, 3, 4 );
SET TrimmedArray = TRIM_ARRAY( UntrimmedArray, 2 );
```

Related Information

ARRAY_AGG Function [Aggregate] [page 247]
ARRAY_MAX_CARDINALITY Function [Composite] [page 248]
CARDINALITY Function [Composite] [page 274]

1.3.2.233 TRUNCNUM Function [Numeric]

Truncates a number at a specified number of places after the decimal point.

Syntax

```
{ TRUNCNUM | TRUNCATE } ( numeric-expression, integer-expression )
```

Parameters

numeric-expression

The number to be truncated. This argument may be of type NUMERIC or DOUBLE.

integer-expression

A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative value specifies the number of significant digits to the left of the decimal point at which to round.

Returns

NUMERIC or DOUBLE

Remarks

You should use the TRUNCNUM function, not the TRUNCATE function, when truncating numbers.

Use of the TRUNCATE function is not recommended because the word truncate is a keyword, and therefore requires you to either set the quoted_identifier option to OFF, or put quotes around the word TRUNCATE.

UltraLite: If any parameter is NULL, the result is NULL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 600:

```
SELECT TRUNCNUM( 655, -2 );
```

The following statement: returns the value 655.340:

```
SELECT TRUNCNUM( 655.348, 2 );
```

Related Information

ROUND Function [Numeric] [page 533]
quoted_identifier Option

1.3.2.234 TSEQUAL function [System] (Deprecated)

Compares two TIMESTAMP values and returns whether they are the same.

Syntax

```
TSEQUAL ( timestamp-expression-1, timestamp-expression-2 )
```

Parameters

timestamp-expression-1

A TIMESTAMP value.

timestamp-expression-2

A TIMESTAMP value.

Returns

BIT

Remarks

Not supported by UltraLite Java edition databases.

The TSEQUAL function can only be used in a WHERE clause and is most commonly used as part of an UPDATE statement.

Although the TSEQUAL function can be used to compare two ordinary TIMESTAMP values, the purpose of TSEQUAL is to determine whether or not a row has been modified by another connection by comparing two special Transact-SQL TIMESTAMP values.

In a single-row UPDATE statement using TSEQUAL, if `timestamp-expression-1` is equal to `timestamp-expression-2` and one of these values refers to a column declared with DEFAULT TIMESTAMP and the other refers to the value of the column when the row was last fetched, then the row has not changed since it was fetched and TSEQUAL returns TRUE. If the row was changed by another user, its timestamp has been modified and the TSEQUAL function returns FALSE. If the TSEQUAL function returns FALSE in this situation, the UPDATE is not performed. The application can determine that no rows were updated by examining the number of rows affected, for example by using @@rowcount. If no rows were affected, the application can assume that the row was modified by another user and that it needs to be re-fetched.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Suppose you create a TIMESTAMP column Products.LastUpdated to store the timestamp for the last time the row was updated. The following example uses the TSEQUAL function to change a row value. An update is applied to the row only when the row has not been changed since it was last fetched.

```
SELECT LastUpdated into old_ts_value
FROM GROUPO.Products
WHERE ID = '300';
```

```
UPDATE GROUPO.Products
SET Color = 'Yellow'
WHERE ID = '300'
AND TSEQUAL( LastUpdated, old_ts_value );
```

Related Information

The Special Transact-SQL TIMESTAMP Column and Data Type
TIMESTAMP Special Value [page 111]
UPDATE Statement [page 1463]

1.3.2.235 UCASE Function [String]

Converts all characters in a string to uppercase.

Syntax

```
UCASE( string-expression )
```

Parameters

string-expression

The string to be converted to uppercase.

Returns

LONG NVARCHAR when used on NCHAR data
LONG VARCHAR when used on CHAR data if the database collation is UCA
Otherwise, the data type is the same as the input data type
UltraLite returns the same data type as the input data type

Remarks

This function is identical to the UPPER function.

Standards

ANSI/ISO SQL Standard

Not in the standard. The UPPER function is ANSI/ISO SQL Standard compliant.

Example

The following statement returns the value CHOCOLATE:

```
SELECT UCASE( 'ChocoLate' );
```

Related Information

String Functions [page 222]
UPPER Function [String] [page 611]
LCASE Function [String] [page 434]

1.3.2.236 UNICODE Function [String]

Returns an integer containing the Unicode code point of the first character in the string, or NULL if the first character is not a valid encoding.

Syntax

```
UNICODE( nchar-string-expression )
```

Parameters

nchar-string-expression

The NCHAR string whose first character is to be converted to an integer.

Returns

INT

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the integer 65536:

```
SELECT UNICODE(UNISTR( '\u010000data' ));
```

Related Information

CONNECTION_EXTENDED_PROPERTY Function [String] [page 290]
NCHAR Function [String] [page 466]
TO_CHAR Function [String] [page 589]
TO_NCHAR Function [String] [page 591]
UNISTR Function [String] [page 609]

1.3.2.237 UNISTR Function [String]

Converts a string containing characters and Unicode escape sequences to an NCHAR string.

Syntax

```
UNISTR( string-expression )
```

Parameters

string-expression

The string to be converted.

Returns

- NVARCHAR
- LONG NVARCHAR

Remarks

The UNISTR function allows the use of Unicode characters that cannot be represented in the CHAR character set used by the SQL statement. For example, in an English environment, the UNISTR function could be used to include Chinese characters.

The UNISTR function offers similar functionality to the N'' constant, however the UNISTR function allows Unicode characters and characters from the CHAR character set, whereas the N'' constant only allows characters from the CHAR character set.

The `string-expression` contains characters and Unicode escape sequences. The Unicode escape sequences are of the form \uXXXX or \uXXXXXX, where each X is a hexadecimal digit. The UNISTR function converts each character and each Unicode escape sequence to the corresponding Unicode character.

If a 6-digit Unicode escape sequence is used, its value must not exceed 10FFFF, the largest Unicode code point. A sequence such as \u234567 is not a 6-digit Unicode escape sequence. It is the 4-digit sequence \u2345 followed by the characters 6 and 7.

If two adjacent Unicode escape sequences form a UTF-16 surrogate pair, they are combined into one Unicode character in the output.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the string Hello:

```
SELECT UNISTR( 'Hel\u006c\u006F' );
```

The following example combines the UTF-16 surrogate pair D800-DF02 into the Unicode code point 10302:

```
SELECT UNISTR( '\uD800\uDF02' );
```

The following example is equivalent to the previous one:

```
SELECT UNISTR( '\u010302' );
```

Related Information

Strings [page 11]
CONNECTION_EXTENDED_PROPERTY Function [String] [page 290]
NCHAR Function [String] [page 466]
TO_CHAR Function [String] [page 589]
TO_NCHAR Function [String] [page 591]
UNICODE Function [String] [page 608]

1.3.2.238 UPPER Function [String]

Converts all characters in a string to uppercase.

Syntax

```
UPPER( string-expression )
```

Parameters

string-expression

The string to be converted to uppercase.

Returns

LONG NVARCHAR when used on NCHAR data
LONG VARCHAR when used on CHAR data if the database collation is UCA
Otherwise, the data type is the same as the input data type
UltraLite returns the same data type as the input data type

Remarks

This function is identical to the UCASE function.

Standards

ANSI/ISO SQL Standard

Core feature.

Example

The following statement returns the value CHOCOLATE:

```
SELECT UPPER( 'ChocoLate' );
```

Related Information

String Functions [page 222]
UCASE Function [String] [page 607]
LCASE Function [String] [page 434]
LOWER Function [String] [page 446]

1.3.2.239 USER_ID Function [System]

Returns the numeric user ID for the specified user name.

Syntax

```
USER_ID( [ user-name ] )
```

Parameters

user-name

The user name for the numeric user ID you are searching for.

Returns

UNSIGNED INTEGER

Remarks

If you do not specify `user-name`, the numeric user ID of the current user is returned.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the numeric user ID for user name GROUPO:

```
SELECT USER_ID( 'GROUPO' );
```

Related Information

USER_NAME Function [System] [page 613]
SUSER_ID Function [System] [page 583]

1.3.2.240 USER_NAME Function [System]

Returns the user name for the specified user ID.

Syntax

```
USER_NAME( [ user-id ] )
```

Parameters

user-id

The user ID of the user you are searching for.

Returns

VARCHAR

Remarks

If you do not specify `user-id`, the user name of the current user is returned.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns GROUPO, the user name for user ID 101:

```
SELECT USER_NAME( 101 );
```

Related Information

USER_ID Function [System] [page 612]
SUSER_NAME Function [System] [page 584]

1.3.2.241 UUIDTOSTR Function [String]

Converts a unique identifier value (UUID, also known as GUID) to a string value.

Syntax

```
UUIDTOSTR( uuid-expression )
```

Parameters

uuid-expression

A unique identifier value.

Returns

VARCHAR

Remarks

Converts a unique identifier to a string value in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, where x is a hexadecimal digit. If the binary value is not a valid uniqueidentifier, NULL is returned.

This function is useful for viewing a UUID value.

Note

In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type was defined as a user-defined data type and the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values.In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type and the database server carries out conversions as needed. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

UltraLite: In databases created before version 9.0.2, the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions..

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a table mytab with two columns. Column pk has a unique identifier data type, and column c1 has an integer data type. It then inserts two rows with the values 1 and 2 respectively into column c1.

```
CREATE TABLE mytab(
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    c1 INT );
INSERT INTO mytab( c1 ) values ( 1 );
INSERT INTO mytab( c1 ) values ( 2 );
```

Executing the following SELECT statement returns all the data in the newly created table:

```
SELECT * FROM mytab;
```

You will see a two-column, two-row table. The value displayed for column pk will be binary values.

To convert the unique identifier values into a readable format, execute the following statement:

```
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

The UUIDTOSTR function is not needed for databases created with version 9.0.2 or later.

Related Information

String Functions [page 222]
UNIQUEIDENTIFIER Data Type [page 183]
NEWID Function [Miscellaneous] [page 467]
STRTOUUID Function [String] [page 575]

1.3.2.242 VAR_POP Function [Aggregate]

Computes the statistical variance of a population consisting of a numeric-expression, as a DOUBLE.

Syntax

Expression

```
VAR_POP( numeric-expression )
```

Window function

```
VAR_POP( numeric-expression ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

numeric-expression

The expression whose population-based variance is calculated over a set of rows. The expression is commonly a column name.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

The population-based variance (s^2) of `numeric-expression` (x) is computed according to the following formula:

$s^2 = (1/N) * SUM(x_I - mean(x))^2$

This variance does not include rows where `numeric-expression` is NULL. It returns NULL for a group containing no rows.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

The VAR_POP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions". When used as window function, VAR_POP comprises part of optional SQL Foundation Feature T611, "Elementary OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the VAR_POP function, combined with an outer reference.

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    VAR_POP( quantity ) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	203.9021...
2000	2	27.050847	225.8109...
...	...	...	...

Related Information

Aggregate Functions [page 208]
Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
WINDOW Clause [page 1489]

1.3.2.243 VAR_SAMP Function [Aggregate]

Computes the statistical variance of a sample consisting of a numeric-expression, as a DOUBLE.

Syntax

Expression

```
VAR_SAMP( numeric-expression )
```

Window function

```
VAR_SAMP( numeric-expression ) OVER ( window-spec )
```

```
window-spec : see the Remarks section below
```

Parameters

numeric-expression

The expression whose sample-based variance is calculated over a set of rows. The expression is commonly a column name.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

The variance (s^2) of `numeric-expression` (x) is computed according to the following formula, which assumes a normal distribution:

$s^2 = (1/(N - 1)) * SUM(x_I - mean(x))^2$

This variance does not include rows where `numeric-expression` is NULL. It returns NULL for a group containing either 0 or 1 rows.

Specifying this function with `window-spec` represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

Standards

ANSI/ISO SQL Standard

The VAR_SAMP function comprises part of optional ANSI/ISO SQL Language Feature T621, "Enhanced numeric functions". When used as window function, VAR_SAMP comprises part of optional ANSI/ISO SQL Foundation Feature T611, "Elementary OLAP operations". The VARIANCE syntax is not in the standard.

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional ANSI/ISO SQL Language Feature F561, "Full value expressions". The software also supports ANSI/ISO SQL Language Feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

The software does not support optional ANSI/ISO SQL Feature F442, "Mixed column references in set functions". The software does not permit the arguments of an aggregate function to include both a column reference from the query block containing the VAR_SAMP function, combined with an outer reference.

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    VAR_SAMP( quantity ) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	205.1158...
2000	2	27.050847	227.0939...
...	...	...	...

Related Information

Aggregate Functions [page 208]
Mathematical Formulas for the Aggregate Functions
Window Functions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
VARIANCE Function [Aggregate] [page 622]
WINDOW Clause [page 1489]

1.3.2.244 VAREXISTS Function [Miscellaneous]

Returns 1 if a user-defined variable exists with the specified name. Returns 0 if no such variable exists.

Syntax

```
VAREXISTS( variable-name-string [, owner ] )
```

Parameters

variable-name-string

The name of the variable to be tested, as a string (for example, 'myVariable').
owner The user ID of the owner of the variable, as a string. `owner` is only for use with owned database-scope variables.

Returns

INT

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following IF statement checks to see if a variable called start_time exists. If it doesn't, then the database server creates a connection-scope variable with that name, and sets its value to the current time.

```
IF VAREXISTS( 'start_time' ) = 0 THEN
    CREATE VARIABLE start_time TIMESTAMP;
END IF;
SET start_time = CURRENT TIMESTAMP;
```

The following IF statement checks to see if a database-scope variable named run_time owned by user ID jsmith exists. If it doesn't, then the database server creates the variable, and sets its value to the current time.

```
IF VAREXISTS( 'run_time', 'jsmith' ) = 0 THEN
    CREATE DATABASE VARIABLE jsmith.run_time TIMESTAMP = CURRENT TIMESTAMP;
END IF;
```

Related Information

CREATE VARIABLE Statement [page 1047]
DECLARE Statement [page 1057]
IF Statement [page 1220]

1.3.2.245 VARIANCE Function [Aggregate]

An alias for VAR_SAMP.

Related Information

VAR_SAMP Function [Aggregate] [page 618]

1.3.2.246 WATCOMSQL Function [Miscellaneous]

Rewrites a Transact-SQL statement in the Watcom SQL dialect. This can be useful when converting existing Adaptive Server Enterprise stored procedures into Watcom SQL syntax.

Syntax

```
WATCOMSQL( sql-statement-string )
```

Parameters

sql-statement-string

The SQL statement that the function rewrites into the Watcom SQL dialect.

Returns

LONG VARCHAR

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the string `'select Surname as last_name from GROUPO.Employees'`:

```
SELECT WATCOMSQL( 'SELECT last_name = Surname FROM GROUPO.Employees' ) FROM
SYS.DUMMY;
```

Related Information

SQLDIALECT Function [Miscellaneous] [page 562]
TRANSACTSQL Function [Miscellaneous] [page 599]

1.3.2.247 WEEKS Function [Date and Time]

Manipulates a TIMESTAMP or returns the number of weeks between two TIMESTAMP values.

Syntax

Returns the number of weeks between 0000-02-29 and a TIMESTAMP value

```
WEEKS( timestamp-expression )
```

Returns the number of weeks between two TIMESTAMP values

```
WEEKS( timestamp-expression, timestamp-expression )
```

Adds weeks to a TIMESTAMP value

```
WEEKS( timestamp-expression, integer-expression )
```

Parameters

timestamp-expression

A date and time value of type TIMESTAMP.

integer-expression

The number of weeks to be added to `timestamp-expression`. If `integer-expression` is negative, the appropriate number of weeks is subtracted from `timestamp-expression`. If you supply an `integer-expression`, `timestamp-expression` must be explicitly cast as a DATE or TIMESTAMP.

Returns

INTEGER when comparing two TIMESTAMP values.

TIMESTAMP when adding weeks to a TIMESTAMP value.

Remarks

Given a single date, the WEEKS function returns the number of weeks since 0000-02-29.

Given two dates, the WEEKS function returns the number of weeks between them. The WEEKS function is similar to the DATEDIFF function, however the method used to calculate the number of weeks between two dates is not the same and can return a different result. The return value for WEEKS is determined by dividing the number of days between the two dates by seven, and then rounding down. However, DATEDIFF uses number of week boundaries in its computation. This can cause the values returned from the two functions to be different. For example, if the first date is a Friday and the second date is the following Monday, the WEEKS function returns a difference of 0, but the DATEDIFF function returns a difference of 1. While neither method is better than the other, you should consider the difference when choosing between WEEKS and DATEDIFF.

Given a date and an integer, the WEEKS function adds the integer number of weeks to `timestamp-expression`. With this syntax, you must explicitly cast `timestamp-expression` as a TIME, DATE, or TIMESTAMP data type. If `timestamp-expression` is a TIME value, the current date is assumed.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 8, signifying that 2008-09-13 10:07:12 is eight weeks after 2008-07-13 06:07:12:

```
SELECT WEEKS( '2008-07-13 06:07:12', '2008-09-13 10:07:12' );
```

The following statement returns the value 104792, signifying that the date is 104792 weeks after 0000-02-29:

```
SELECT WEEKS( '2008-07-13 06:07:12' );
```

The following statement returns the TIMESTAMP value 2008-06-16 21:05:07.0, indicating the date and time five weeks after 2008-05-12 21:05:07:

```
SELECT WEEKS( CAST( '2008-05-12 21:05:07' AS TIMESTAMP ), 5);
```

Related Information

DATEDIFF Function [Date and Time] [page 317]
DATEADD Function [Date and Time] [page 316]
CAST Function [Data Type Conversion] [page 275]

1.3.2.248 WRITE_CLIENT_FILE Function [String]

Creates and writes to a file on the client computer.

Syntax

```
WRITE_CLIENT_FILE( filename , blob-expression [ , 'A' ] )
```

Parameters

filename

The name of the file on the client computer. The name is resolved on the client computer relative to the current working directory of the client application.

blob-expression

A binary string to be written to `filename` on the client computer.

A

By default, if the file already exists, it is overwritten. If you want the data to be appended to existing data, specify 'A'. If the file does not already exist, and you specify 'A', the file is still created.

Returns

INT

Remarks

The database server converts `filename` from the database character set to the client character set. On the client computer, `filename` is then converted to the operating system character set.

Since the data is a binary string, if you want the data to be in a particular character set, or compressed, or encrypted, you must perform these operations on the data before sending it to the WRITE_CLIENT_FILE function.

Writing of the file is performed by the client software library and the transfer of data is done by using the **Command Sequence** (**CmdSeq**) communication protocol. Client-side data transfers are not supported by the **Tabular Data Stream** (**TDS**) protocol.

Privileges

When writing to a file on a client computer:

- You must have the WRITE CLIENT FILE system privilege.
- The client application must have write permissions on the computer being written to.
- The allow_write_client_file database option must be enabled.
- The WRITE_CLIENT_FILE feature must be enabled.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Access to Data on Client Computers
allow_write_client_file Option
-sf Database Server Option
UNLOAD Statement [page 1453]
CSCONVERT Function [String] [page 309]
DECOMPRESS Function [String] [page 339]
DECRYPT Function [String] [page 340]

1.3.2.249 XMLAGG Function [Aggregate]

Generates a forest of XML elements from a collection of XML values.

Syntax

```
XMLAGG( expression [ ORDER BY order-by-expression ] )
```

Parameters

expression

An XML value. The content is escaped unless the data type is XML. The `order-by-expression` orders the elements returned by the function.

order-by-expression

An expression used to order the XML elements according to the value of this expression.

When an ORDER BY clause contains constants, they are interpreted by the optimizer and then replaced by an equivalent ORDER BY clause. For example, the optimizer interprets ORDER BY 'a' as ORDER BY expression.

A query block containing more than one aggregate function with valid ORDER BY clauses can be executed if the ORDER BY clauses can be logically combined into a single ORDER BY clause. For example, the following clauses:

```
ORDER BY expression1, 'a', expression2
```

```
ORDER BY expression1, 'b', expression2, 'c', expression3
```

are subsumed by the clause:

```
ORDER BY expression1, expression2, expression3
```

Returns

XML

Remarks

Any values that are NULL are omitted from the result. If all inputs are NULL, or there are no rows, the result is NULL. If you require a well-formed XML document, you must ensure that your query is written so that the generated XML has a single root element.

Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format when you execute a query that contains XMLAGG.

Standards

ANSI/ISO SQL Standard

XMLAGG is part of optional ANSI/ISO SQL Language Feature X034. The optional ORDER BY clause for the XMLAGG function comprises optional ANSI/ISO SQL Language Feature X035.

Example

The following statement generates an XML document that shows the orders placed by each customer:

```
SELECT XMLELEMENT( NAME "order",
                   XMLATTRIBUTES( ID AS order_id ),
                     ( SELECT XMLAGG(
                         XMLELEMENT(
                           NAME "Products",
                           XMLATTRIBUTES( ProductID, Quantity AS
"quantity_shipped" ) ) )
                       FROM SalesOrderItems soi
                       WHERE soi.ID = so.ID
                     )
                   ) AS products_ordered
FROM GROUPO.SalesOrders so
ORDER BY so.ID;
```

Related Information

Use of the XMLAGG Function

1.3.2.250 XMLCONCAT Function [String]

Produces a forest of XML elements.

Syntax

```
XMLCONCAT( xml-value [, ... ] )
```

Parameters

xml-value

The XML values to be concatenated.

Returns

XML

Remarks

Generates a forest of XML elements. In an unparsed XML document, a forest refers to the multiple root nodes within the document. NULL values are omitted from the result. If all the values are NULL, then NULL is returned. The XMLCONCAT function does not check whether the argument has a prolog. If you require a well-formed XML document, you must ensure that your query is written so that a single root element is generated.

Element content is always escaped unless the data type is XML. Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format when you execute a query that contains a XMLCONCAT function.

Standards

ANSI/ISO SQL Standard

XMLCONCAT comprises part of the optional ANSI/ISO SQL Language Feature X020.

Example

The following query generates <CustomerID>, <cust_fname>, and <cust_lname> elements for each customer:

```
SELECT XMLCONCAT( XMLELEMENT ( NAME CustomerID, ID ),
                  XMLELEMENT( NAME cust_fname, GivenName ),
                  XMLELEMENT( NAME cust_lname, Surname )
                ) AS "Customer Information"
FROM GROUPO.Customers
WHERE ID < 120;
```

Related Information

Use of the XMLCONCAT Function
String Functions [page 222]
XMLELEMENT Function [String] [page 629]
XMLFOREST Function [String] [page 632]

1.3.2.251 XMLELEMENT Function [String]

Produces an XML element within a query.

Syntax

```
XMLELEMENT( { NAME element-name-expression } | string-expression
```

```
    [, XMLATTRIBUTES ( attribute-value-expression [ AS attribute-name ] , ... ) ]
    [, element-content-expression , ... ]
)
```

Parameters

element-name-expression

An identifier. For each row, an XML element with the same name as the identifier is generated.

attribute-value-expression

An attribute of the element. This optional argument allows you to specify an attribute value for the generated element. This argument specifies the attribute name and content. If the `attribute-value-expression` is a column name, then the attribute name defaults to the column name. You can change the attribute name by specifying the `attribute-name argument.`

You can specify a variable name for `attribute-value-expression`.

element-content-expression

The content of the element. This can be any string expression. You can specify an unlimited number of `element-content-expression` arguments and they are concatenated together. For example, the following SELECT statement returns the value <x>abcdef</x>:

```
SELECT XMLELEMENT( NAME x, 'abc', 'def' );
```

Returns

XML

Remarks

NULL element values and NULL attribute values are omitted from the result. The letter case for both element and attribute names is taken from the query.

Element content is always escaped unless the data type is XML. Invalid element and attribute names are also quoted. For example, consider the following statement:

```
SELECT XMLELEMENT('H1', f_get_page_heading() );
```

If the function f_get_page_heading is defined as RETURNS LONG VARCHAR or RETURNS VARCHAR(1000), then the result is HTML encoded:

```
CREATE FUNCTION f_get_page_heading() RETURNS LONG VARCHAR
   BEGIN
      RETURN ('<B>My Heading</B>');
```

```
    END;
```

The above SELECT statement returns the following:

```
<H1>&lt;B&gt;My Heading&lt;/B&gt;</H1>
```

If the function is declared as RETURNS XML, then the above SELECT statement returns the following:

```
<H1><B>My Heading</B></H1>
```

XMLELEMENT functions can be nested to create a hierarchy. To return different elements at the same level of the document hierarchy, use the XMLFOREST function.

Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format when you execute a query that contains the XMLELEMENT function.

Standards

ANSI/ISO SQL Standard

XMLELEMENT constitutes part of optional ANSI/ISO SQL Language Feature X031. Omitting the NAME keyword and using a string expression as the first argument is not in the standard. The software does not support the optional OPTION clause with the XMLELEMENT function.

Example

The following example produces an <item_name> element for each product in the result set, where the product name is the content of the element:

```
SELECT ID, XMLELEMENT( NAME item_name, p.Name )
FROM Products p
WHERE ID > 400;
```

The following example returns `<A HREF="http://www.sap.com/" TARGET="_top">SAP web site</A>`:

```
SELECT XMLELEMENT(
   'A',
   XMLATTRIBUTES( 'http://www.sap.com/'
      AS "HREF", '_top' AS "TARGET"),
   'SAP web site'
);
```

The following example returns `<table><tbody><tr align="center" valign="top"><td>Cell 1 info</td><td>Cell 2 info</td></tr></tbody></table>`:

```
SELECT XMLELEMENT( name "table",
         XMLELEMENT( name "tbody",
           XMLELEMENT( name "tr",
             XMLATTRIBUTES('center' AS "align", 'top' AS "valign"),
             XMLELEMENT( name "td", 'Cell 1 info' ),
             XMLELEMENT( name "td", 'Cell 2 info' )
           )
```

```
          )
        );
```

The following example returns '<x>abcdef</x>','<custom_element>abcdef</custom_element>':

```
CREATE VARIABLE @my_element_name VARCHAR(200);
SET @my_element_name = 'custom_element';
SELECT XMLELEMENT( NAME x, 'abc', 'def' ),
   XMLELEMENT( @my_element_name,'abc', 'def' );
```

Related Information

Use of SQL/XML to Obtain Query Results as XML
Use of the XMLELEMENT Function
String Functions [page 222]
XMLCONCAT Function [String] [page 628]
XMLFOREST Function [String] [page 632]

1.3.2.252 XMLFOREST Function [String]

Generates a forest of XML elements.

Syntax

```
XMLFOREST( element-content-expression [ AS element-name ] , ... )
```

Parameters

element-content-expression

A string. An element is generated for each `element-content-expression` argument that is specified. The `element-content-expression` value becomes the content of the element. For example, if you specify the EmployeeID column from the Employees table for this argument, then an <EmployeeID> element containing an EmployeeID value is generated for each value in the table.

Specify the `element-name` argument to assign a name other than the `element-content-expression` to the element, otherwise the element name defaults to the `element-content-expression` name.

Returns

XML

Remarks

Produces a forest of XML elements. In the unparsed XML document, a forest refers to the multiple root nodes within the document. When all the arguments to the XMLFOREST function are NULL, a NULL value is returned. If only some values are NULL, the NULL values are omitted from the result. Element content is always quoted unless the data type is XML. You cannot specify attributes using the XMLFOREST function. Use the XMLELEMENT function to specify attributes for generated elements.

Element names are escaped unless the data type is XML.

If you require a well-formed XML document, you must ensure that your query is written so that a single root element is generated.

Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format when you execute a query that contains XMLFOREST.

Standards

ANSI/ISO SQL Standard

XMLFOREST constitutes part of optional ANSI/ISO SQL Language Feature X032. The software does not support the optional XMLNAMESPACES clause, or the OPTION clause, with the XMLFOREST function.

Example

The following statement produces an XML element for the first and last name of each employee:

```
SELECT EmployeeID,
       XMLFOREST( GivenName, Surname )
       AS "Employee Name"
FROM Employees;
```

Related Information

Use of the XMLFOREST Function
String Functions [page 222]
XMLELEMENT Function [String] [page 629]
XMLCONCAT Function [String] [page 628]

1.3.2.253 XMLGEN Function [String]

Generates an XML value based on an XQuery constructor.

Syntax

```
XMLGEN( xquery-constructor , content-expression [ AS variable-name ] , ... )
```

Parameters

xquery-constructor

An XQuery constructor. The XQuery constructor is an item defined in the XQuery language. It gives a syntax for constructing XML elements based on XQuery expressions. The `xquery-constructor` argument must be a well-formed XML document with one or more variable references. A variable reference is enclosed in curly braces and must be prefixed with a $ and have no surrounding white space. For example:

```
SELECT XMLGEN( '<a>{$x}</a>', 1 AS x );
```

content-expression

A variable. You can specify multiple `content-expression` arguments. The optional `variable-name` argument is used to name the variable. For example:

```
SELECT XMLGEN( '<emp EmployeeID="{$EmployeeID}"><StartDate>{$x}</StartDate></
emp>',
               EmployeeID, StartDate
               AS x )
FROM GROUPO.Employees;
```

Returns

XML

Remarks

Computed constructors as defined in the XQuery specification are not supported by the XMLGEN function.

When you execute a query that contains an XMLGEN function, data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format.

Element content is always escaped unless the data type is XML. Illegal XML element and attribute names are also escaped.

Standards

ANSI/ISO SQL Standard

Not in the standard. XMLGEN provides similar functionality to the ANSI SQL XMLDOCUMENT function.

Example

The following example generates <emp>, <Surname>, <GivenName>, and <StartDate> elements for each employee:

```
SELECT XMLGEN( '<emp EmployeeID="{$EmployeeID}">
                  <Surname>="{$Surname}"</Surname>
                  <GivenName>="{$GivenName}"</GivenName>
                  <StartDate>="{$StartDate}"</StartDate>
               </emp>',
               EmployeeID,
               Surname,
               GivenName,
               StartDate
             ) AS employee_list
FROM GROUPO.Employees;
```

Related Information

Use of SQL/XML to Obtain Query Results as XML
Use of the XMLGEN Function
String Functions [page 222]

1.3.2.254 YEAR Function [Date and Time]

Returns the year component of the TIMESTAMP argument.

Syntax

```
YEAR( timestamp-expression )
```

Parameters

timestamp-expression

A TIMESTAMP value.

Returns

SMALLINT

Remarks

The value returned is the years component of the given TIMESTAMP value, returned as a SMALLINT.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example returns the value 2001:

```
SELECT YEAR( '2001-09-12' );
```

1.3.2.255 YEARS Function [Date and Time]

Manipulates a TIMESTAMP or returns the number of years between two TIMESTAMP values.

Syntax

Return the number of years between year 0000 and a TIMESTAMP value

```
YEARS( timestamp-expression )
```

Return the number of years between two TIMESTAMP values

```
YEARS( timestamp-expression, timestamp-expression )
```

Add years to a TIMESTAMP value

```
YEARS( timestamp-expression, integer-expression )
```

Parameters

timestamp-expression

A date and time value of type TIMESTAMP.

integer-expression

The number of years (as a SMALLINT value) to be added to `timestamp-expression`. If `integer-expression` is negative, the appropriate number of years is subtracted from `timestamp-expression`. If you supply an `integer-expression`, the `timestamp-expression` must be explicitly cast as a DATE, TIME, or TIMESTAMP value. If `timestamp-expression` is a TIME, the current year is assumed.

Returns

SMALLINT when comparing two TIMESTAMP values.

TIMESTAMP when adding years to a TIMESTAMP value.

Remarks

The value of YEARS is computed by counting the number of first days of the year between the two dates.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements both return -4:

```
SELECT YEARS( '1998-07-13 06:07:12',
              '1994-03-13 08:07:13' );
```

```
SELECT DATEDIFF( year,
   '1998-07-13 06:07:12',
   '1994-03-13 08:07:13' );
```

The following statements return 1998:

```
SELECT YEARS( '1998-07-13 06:07:12' )
SELECT DATEPART( year, '1998-07-13 06:07:12' );
```

The following statements return the given date advanced 300 years:

```
SELECT YEARS( CAST( '1998-07-13 06:07:12' AS TIMESTAMP ), 300 )
```

```
SELECT DATEADD( year, 300, '1998-07-13 06:07:12' );
```

Related Information

DATEDIFF Function [Date and Time] [page 317]
DATEADD Function [Date and Time] [page 316]
CAST Function [Data Type Conversion] [page 275]

1.3.2.256 YMD Function [Date and Time]

Returns a date value corresponding to the given year, month, and day of the month. Arguments are INTEGER values from -32768 to 32767.

Syntax

```
YMD( smallint-expression1, smallint-expression2, smallint-expression3 )
```

Parameters

smallint-expression1

The year.

smallint-expression2

The number of the month. The year is adjusted if the month is outside the range 1-12.

smallint-expression3

The day number. The day can be any integer; the date is adjusted accordingly.

Returns

DATE

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the value 1998-06-12:

```
SELECT YMD( 1998, 06, 12 );
```

If the values are outside their normal range, the date is adjusted accordingly. For example, the following statement returns the DATE value 2000-03-01:

```
SELECT YMD( 1999, 15, 1 );
```

1.4 SQL Statements

There are several conventions used in the SQL statement documentation.

In this section:

1.4.1 Common Elements in SQL Syntax

Learn about language elements that are found in the syntax of many SQL statements.

column-name

An identifier that represents the name of a column.

condition

An expression that evaluates to TRUE, FALSE, or UNKNOWN.

connection-name

A string representing the name of an active connection.

data-type

A storage data type.

expression

An expression. A common example of an expression in syntax is a column name.

filename

A string containing a file name.

hostvar

A C language variable, declared as a host variable preceded by a colon.

materialized-view-name

An identifier that represents the name of a materialized view.

number

Any sequence of digits followed by an optional decimal part and preceded by an optional negative sign. Optionally, the number can be followed by an E and then an exponent. For example:

```
42
-4.038
.001
3.4e10
1e-10
```

owner

An identifier representing the user ID who owns a database object.

query-block

A query block is a simple query expression, or a query expression with an ORDER BY clause.

query-expression

A query expression can be a SELECT, UNION, INTERSECT, or EXCEPT block (that is, a statement that does not contain an ORDER BY, WITH, FOR, FOR XML, or OPTION clause), or any combination of such blocks.

role-name

An identifier representing the role name of a foreign key. In conceptual database modeling, a verb or phrase that describes a relationship from one point of view. You can describe each relationship with two roles. Examples of roles are "contains" and "is a member of."

savepoint-name

An identifier that represents the name of a savepoint.

search-condition

A condition that evaluates to TRUE, FALSE, or UNKNOWN.

special-value

A special value.

statement-label

An identifier that represents the label of a loop or compound statement.

statement-list

A list of SQL statements, each ending with a semicolon.

string-expression

An expression that resolves to a string.

table-list

A list of table names, which may include correlation names.

table-name

An identifier that represents the name of a table.

userid

An identifier representing a user name.

variable-name

An identifier that represents a variable name.

window-name

An identifier that represents a window name. Used in syntax related to window definition (for example, the WINDOW clause, and window functions such as RANK).

Related Information

Savepoints Within Transactions
SQL Variables [page 123]
Materialized Views
Host Variables in Embedded SQL
Strings [page 11]
Database Connections
Control Statements
Special Values [page 87]
Key Joins
Truth Value Search Conditions [page 84]
Expressions in SQL Statements [page 34]
SQL Data Types [page 129]
Search Conditions [page 58]
FROM Clause [page 1173]
Identifiers [page 6]

1.4.2 Syntax Conventions

These are the conventions used in the SQL syntax descriptions.

Keywords

All SQL keywords appear in uppercase, like the SQL statement ALTER TABLE in the following example:

```
ALTER TABLE [owner.]table-name
```

Placeholders (also called variable values, or variables)

Items that must be replaced with appropriate identifiers or expressions appear in italics, like the words `owner` and `table-name` in the following example:

```
ALTER TABLE [owner.]table-name
```

When placeholders are multiple words, the words are separated by hyphens (for example, `table-name`) instead of underscores to differentiate them from parameters.

Parameters

Parameters for objects like functions are set in the software, and when specified, must match the wording used in the software. When parameters are multiple words, the words are separated by underscores. Parameters in syntax documentation are bolded (for example, **table_name**).

Clause order

If the order of optional clauses is significant in SQL statement syntax, the clauses are already listed in the syntax in the order in which they should be listed. As a best practice, and to aid you in finding specific clause help for a statement, follow the order of clauses prescribed in the documentation.

```
CREATE SYNCHRONIZATION SUBSCRIPTION [ subscription-name ]
TO publication-name
[ FOR ml-username, ... ]
...
```

Optional portions

Optional portions of a statement are enclosed by square brackets. For example:

```
RELEASE SAVEPOINT [ savepoint-name ]
```

These square brackets indicate that specifying a value for `savepoint-name` is optional. The square brackets should not be typed.

You might also see square brackets around a portions of keywords. For example, the following syntax indicates that you can use either COMMIT TRAN or COMMIT TRANSACTION:

```
COMMIT TRAN[SACTION] ...
```

Likewise, the following syntax indicates that you can use either COMMIT or COMMIT WORK:

```
COMMIT [ WORK ]
```

Denoting the repetition of a part of syntax

An item that can be repeated is denoted using an ellipsis (three dots) and corresponding bracketing that confirms the optional repetition. Both of the following styles are used in syntax documentation and are semantically equivalent, but one style may be used over the other to aid readability in complicated syntaxes.

```
ADD column-definition [ column-constraint, ... ]
```

```
ADD column-definition [ column-constraint [,...] ]
```

Sets of values

When only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in brackets (braces or square brackets).

```
{ ON | OFF }
[ ASC | DESC ]
```

In the first example, one value is required, and can be either ON or OFF. In the second example, you can specify one of ASC, DESC, or neither. As always, brackets should not be typed unless they are bolded and part of the string you are required to type.

Alternatives

When precisely one of the options must be chosen, the alternatives are enclosed in braces.

```
[ QUOTES { ON | OFF } ]
```

In this case, if the QUOTES option is chosen, one of ON or OFF must be provided. The brackets and braces should not be typed.

Bracketing

Braces (**{ }**) indicate a set that you must choose a value from and are almost never typed unless specifically identified as such. Square brackets (**[ ]**) indicate that the content within the brackets is optional; square brackets used in illustrating syntax are never typed. Round brackets/parentheses (**( )**) are part of syntax and must be typed.

Square brackets can be used in SQL syntax as an alternative to double quotation marks, for example, to avoid having an identifier interpreted as a keyword. In the following example, the square brackets are a quoting mechanism and do not mean that the word is optional.

```
SELECT [option] FROM SYS.SYSOPTION;
```

1.4.3 Statement Applicability Indicators

Some statement titles are followed by an indicator in square brackets that indicate where the statement can be used.

These indicators are as follows:

[ESQL]

The statement is for use in Embedded SQL.

[Interactive SQL]

The statement can be used only in Interactive SQL.

[SP]

The statement is for use in stored procedures, triggers, or batches.

[T-SQL]

The statement is implemented for compatibility with Adaptive Server Enterprise. Sometimes the statement cannot be used in stored procedures that are not in Transact-SQL format. In other cases, an

alternative statement closer to the ANSI/ISO SQL Standard is recommended unless Transact-SQL compatibility is an issue.

[External call]

The statement is for use in calling external functions and procedures.

[MobiLink]

The statement is for use only in MobiLink clients.

[SQL Remote]

The statement can be used only in SQL Remote.

[Web Service]

The statement is for use in web services clients.

If two sets of brackets are used, the statement can be used in both environments. For example, [ESQL][SP] means a statement can be used in both Embedded SQL and stored procedures.

1.4.4 Alphabetical List of SQL Statements

There are many SQL statements supported by the software.

In this section:

Alters a text configuration object.

ALTER TEXT INDEX Statement [page 762]
Alters the definition of a text index.

ALTER TIME ZONE Statement [page 764]
Modifies a time zone object.

ALTER TRACE EVENT SESSION Statement [page 765]
Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

ALTER TRIGGER Statement [page 768]
Replaces a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.

ALTER USER Statement [page 769]
Alters user settings.

ALTER VIEW Statement [page 772]
Replaces a view definition with a modified version.

ATTACH TRACING Statement (Deprecated) [page 775]
The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The ATTACH TRACING statement starts a diagnostic tracing session (starts sending diagnostic information to the diagnostic tables).

BACKUP DATABASE Statement [page 778]
Backs up a database and transaction log.

BEGIN Statement [page 784]
Specifies a compound statement.

BEGIN Statement [T-SQL] [page 787]
Specifies a compound statement.

BEGIN PARALLEL WORK Statement [page 789]
Saves time when executing a list of CREATE INDEX or LOAD TABLE statements on a computer with multiple logical processors.

BEGIN SNAPSHOT Statement [page 791]
Starts a snapshot at a specified period in time for use with snapshot isolation transactions.

BEGIN TRANSACTION Statement [T-SQL] [page 792]
Begins a user-defined transaction.

BREAK Statement [T-SQL] [page 794]
Exits a compound statement or loop.

CALL Statement [page 795]
Invokes a procedure.

CASE Statement [page 800]
Selects an execution path based on multiple cases.

CASE Statement [T-SQL] [page 802]
Selects an execution path based on multiple cases.

CHECKPOINT Statement [page 805]
Checkpoints the database.

CLEAR Statement [Interactive SQL] [page 806]
Closes any open result sets in Interactive SQL.

CLOSE statement [ESQL] [SP] [page 807]
Closes a cursor.

COMMENT Statement [page 808]
Stores a comment for a database object in the system tables.

COMMIT Statement [page 811]
Makes changes to the database permanent, or terminates a user-defined transaction.

CONFIGURE Statement [Interactive SQL] [page 814]
Opens the Interactive SQL *Options* window.

CONNECT Statement [ESQL] [Interactive SQL] [page 815]
Establishes a connection to a database.

CONTINUE Statement [page 818]
Restarts a loop.

CREATE CERTIFICATE Statement [page 820]
Adds or replaces a certificate in the database from the given file or string. To create a certificate, use the Certificate Creation utility (createcert).

CREATE DATABASE Statement [page 821]
Creates a database.

CREATE DBSPACE Statement [page 830]
Defines a new database space and creates the associated database file.

CREATE DECRYPTED DATABASE Statement [page 832]
Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.

CREATE DECRYPTED FILE Statement [page 834]
Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

CREATE DOMAIN Statement [page 836]
Creates a domain in a database.

CREATE ENCRYPTED DATABASE Statement [page 840]
Creates an obfuscated or encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table obfuscation or encryption enabled.

CREATE ENCRYPTED FILE Statement [page 843]
Creates a strongly encrypted copy of a database file when you cannot use the CREATE ENCRYPTED DATABASE statement. Also creates encrypted copies of the transaction log, transaction log mirror, and dbspace files.

CREATE EVENT Statement [page 847]
Defines an event and its associated handler for automating predefined actions, and to define scheduled actions.

CREATE EXISTING TABLE Statement [page 854]
Creates a new proxy table, which represents an existing object on a remote server.

CREATE EXTERNLOGIN Statement [page 858]
Assigns an alternate login name and password to be used when communicating with a remote server.

CREATE FUNCTION Statement [page 861]
Creates a user-defined SQL function in the database.

CREATE FUNCTION Statement [External Call] [page 866]
Creates an interface to a native or external function.

CREATE FUNCTION Statement [Web Service] [page 874]
Creates a web client function that makes an HTTP or SOAP over HTTP request.

CREATE INDEX Statement [page 886]
Creates an index on a specified table or materialized view.

CREATE LDAP SERVER Statement [page 891]
Creates an LDAP server configuration object.

CREATE LOGIN POLICY Statement [page 894]
Creates a login policy.

CREATE MATERIALIZED VIEW Statement [page 896]
Creates a materialized view.

CREATE MESSAGE Statement [T-SQL] [page 899]
Creates a message number/message string pair. The message number can be used in PRINT and RAISERROR statements.

CREATE MIRROR SERVER Statement [page 901]
Creates or replaces a mirror server that is being used for database mirroring or read-only scale-out.

CREATE MUTEX Statement [page 906]
Creates or replaces a mutex (lock) that can be used to lock a resource such as a file or a procedure.

CREATE ODATA PRODUCER Statement [page 908]
Creates or replaces an OData Producer.

CREATE PROCEDURE Statement [page 913]
Creates a user-defined SQL procedure in the database.

CREATE PROCEDURE Statement [External Call] [page 921]
Creates an interface to a native or external procedure.

CREATE PROCEDURE Statement [Web Service] [page 931]
Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

CREATE PROCEDURE Statement [T-SQL] [page 944]
Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

CREATE PUBLICATION Statement [MobiLink] [SQL Remote] [page 946]
Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

CREATE REMOTE [MESSAGE] Statement [SQL Remote] [page 950]
Identifies a message link and return address for outgoing messages from a database.

CREATE ROLE Statement [page 952]
Creates or replaces a role, creates a user-extended role, or modifies administrators for a role.

CREATE SCHEMA Statement [page 955]
Creates a collection of tables and views for a database user.

CREATE SEMAPHORE Statement [page 957]

Creates or replaces a semaphore and establishes the initial value for its counter. A semaphore is a locking mechanism that uses a counter to communicate and control the availability of a resource such as an external library or procedure.

CREATE SEQUENCE Statement [page 959]
Creates a sequence that can be used to generate primary key values that are unique across multiple tables, and for generating default values for a table.

CREATE SERVER Statement [page 962]
Creates a remote server or a directory access server.

CREATE SERVICE Statement [HTTP Web Service] [page 969]
Creates a new HTTP web service.

CREATE SERVICE Statement [SOAP Web Service] [page 975]
Creates a new SOAP over HTTP or DISH service.

CREATE SPATIAL REFERENCE SYSTEM Statement [page 981]
Creates or replaces a spatial reference system.

CREATE SPATIAL UNIT OF MEASURE Statement [page 989]
Creates or replaces a spatial unit of measurement.

CREATE STATISTICS Statement [page 991]
Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

CREATE SUBSCRIPTION Statement [SQL Remote] [page 993]
Creates a subscription for a remote user to a publication.

CREATE SYNCHRONIZATION PROFILE Statement [MobiLink] [page 996]
Creates a SQL Anywhere synchronization profile.

CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 997]
Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

CREATE SYNCHRONIZATION USER Statement [MobiLink] [page 1000]
Creates a MobiLink user in a SQL Anywhere remote database.

CREATE TABLE Statement [page 1002]
Creates a new table in the database and, optionally, creates a table on a remote server.

CREATE TEMPORARY TRACE EVENT Statement [page 1022]
Creates a user trace event that persists until the database is stopped.

CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
Creates a user trace event session.

CREATE TEXT CONFIGURATION Statement [page 1028]
Creates a text configuration object for use with building and updating text indexes.

CREATE TEXT INDEX Statement [page 1030]
Creates a text index.

CREATE TIME ZONE Statement [page 1033]
Creates a time zone object that can be used by the database to simulate a time zone that is different from the server's time zone.

CREATE TRIGGER Statement [page 1036]

Creates a trigger on a table.

CREATE TRIGGER Statement [T-SQL] [page 1043]
Creates a new trigger in the database in a manner compatible with Adaptive Server Enterprise.

CREATE USER Statement [page 1045]
Creates a database user or group.

CREATE VARIABLE Statement [page 1047]
Creates a connection- or database-scope variable.

CREATE VIEW Statement [page 1051]
Creates a view on the database.

DEALLOCATE DESCRIPTOR Statement [ESQL] [page 1054]
Frees memory associated with a SQL descriptor area.

DEALLOCATE Statement [page 1055]
This statement has no effect in SQL Anywhere, and is ignored. It is provided for compatibility with Adaptive Server Enterprise and Microsoft SQL Server. Refer to your Adaptive Server Enterprise or Microsoft SQL Server documentation for more information about this statement.

Declaration Section [ESQL] [page 1056]
Declares host variables in an Embedded SQL program. Host variables are used to exchange data with the database.

DECLARE Statement [page 1057]
Declares a SQL variable (connection-scope) or an exception within a compound statement (BEGIN...END).

DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
Declares a cursor.

DECLARE LOCAL TEMPORARY TABLE Statement [page 1066]
Declares a local temporary table.

DELETE Statement [page 1070]
Deletes rows from the database.

DELETE Statement (Positioned) [ESQL] [SP] [page 1074]
Deletes the data at the current location of a cursor.

DESCRIBE Statement [ESQL] [page 1076]
Gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

DESCRIBE Statement [Interactive SQL] [page 1080]
Returns information about a given database object.

DETACH TRACING Statement (Deprecated) [page 1084]
The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The DETACH TRACING session ends a diagnostic tracing session.

DISCONNECT Statement [ESQL] [Interactive SQL] [page 1085]
Drops a connection to a database.

DROP CERTIFICATE Statement [page 1087]
Drops a certificate from the database.

DROP CONNECTION Statement [page 1088]

Drops a user's connection to the database.

DROP DATABASE Statement [page 1089]
Deletes all database files associated with a database.

DROP DATATYPE Statement [page 1091]
Removes a data type from the database.

DROP DBSPACE Statement [page 1092]
Removes a dbspace from the database.

DROP DOMAIN Statement [page 1093]
Removes a domain (data type) from the database.

DROP EVENT Statement [page 1094]
Drops an event from the database.

DROP EXTERNLOGIN Statement [page 1095]
Drops an external login from the database.

DROP FUNCTION Statement [page 1097]
Removes a function from the database.

DROP INDEX Statement [page 1098]
Removes an index from the database.

DROP LDAP SERVER Statement [page 1100]
Drops an LDAP server configuration object.

DROP LOGIN POLICY Statement [page 1101]
Drops a login policy.

DROP MATERIALIZED VIEW Statement [page 1103]
Removes a materialized view from the database.

DROP MESSAGE Statement [page 1104]
Removes a message from the database.

DROP MIRROR SERVER Statement [page 1105]
Drops a mirror server.

DROP MUTEX Statement [page 1107]
Drops the specified mutex.

DROP ODATA PRODUCER Statement [page 1108]
Drops an OData Producer.

DROP PROCEDURE Statement [page 1109]
Removes a procedure from the database.

DROP PUBLICATION Statement [MobiLink] [SQL Remote] [page 1111]
Drops a publication.

DROP REMOTE CONNECTION Statement [page 1112]
Drops remote data access connections to a remote server.

DROP REMOTE [MESSAGE] Statement [SQL Remote] [page 1114]
Deletes a message type definition from a database.

DROP ROLE Statement [page 1115]
Removes a role from the database, or converts a user-extended role back to a regular user.

LOAD STATISTICS Statement [page 1246]
Intended for internal use, this statement is used by the dbunload utility to unload column statistics from the old database into the ISYSCOLSTAT system table.

LOAD TABLE Statement [page 1247]
Imports bulk data into a database table from an external file.

LOCK FEATURE Statement [page 1264]
Prevents other concurrent connections from using a database server feature.

LOCK MUTEX Statement [page 1265]
Locks a resource such as a file or system procedure using a predefined mutex.

LOCK TABLE Statement [page 1267]
Prevents other concurrent transactions from accessing or modifying a table.

LOOP Statement [page 1269]
Repeats the execution of a statement list.

MERGE Statement [page 1271]
Merges tables, views, and procedure results into a table or view.

MESSAGE Statement [page 1278]
Displays a message.

NOTIFY SEMAPHORE Statement [page 1282]
Increments the counter associated with a semaphore.

NOTIFY TRACE EVENT Statement [page 1284]
Logs a user-defined trace event to a trace session.

OPEN Statement [ESQL] [SP] [page 1286]
Opens a previously declared cursor to access information from the database.

OUTPUT Statement [Interactive SQL] [page 1289]
Outputs the current query results to a file.

PARAMETERS Statement [Interactive SQL] [page 1296]
Specifies parameters to an Interactive SQL script file.

PASSTHROUGH Statement [SQL Remote] [page 1299]
Starts or stops passthrough mode for SQL Remote administration.

PIVOT Clause [page 1300]
Pivots a table expression in the FROM clause of a SELECT statement (FROM `pivoted-derived-table`) into a pivoted derived table. Pivoted derived tables offer an easy way to rotate row values from a column in a table expression into multiple columns, and perform aggregation where needed on the columns included in the result set.

PREPARE Statement [ESQL] [page 1308]
Prepares a statement to be executed later, or defines a cursor.

PREPARE TO COMMIT Statement [page 1310]
Checks whether a COMMIT can be performed successfully.

PRINT Statement [T-SQL] [page 1312]
Returns a message to the client, or display a message in the database server messages window.

PUT Statement [ESQL] [page 1313]
Inserts a row into the specified cursor.

Signals an exception condition.

START DATABASE Statement [page 1407]
Starts a database on the current database server.

START EXTERNAL ENVIRONMENT Statement [page 1410]
Starts an external environment.

START JAVA Statement [page 1412]
Starts the Java VM.

START LOGGING Statement [Interactive SQL] [page 1413]
Starts logging executed SQL statements and messages to a log file.

START SERVER Statement [Interactive SQL] [page 1415]
Starts a database server.

START SUBSCRIPTION Statement [SQL Remote] [page 1416]
Starts a subscription for a user to a publication.

START SYNCHRONIZATION DELETE Statement [MobiLink] [page 1418]
Restarts logging of deletes for MobiLink synchronization.

START SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink] [page 1419]
Starts a MobiLink synchronization schema change.

STOP DATABASE Statement [page 1421]
Stops a database on the current database server.

STOP EXTERNAL ENVIRONMENT Statement [page 1423]
Stops an external environment.

STOP JAVA Statement [page 1424]
Stops the Java VM.

STOP LOGGING Statement [Interactive SQL] [page 1426]
Stops logging of SQL statements and messages for the current session.

STOP SERVER Statement [page 1427]
Stops a database server.

STOP SUBSCRIPTION Statement [SQL Remote] [page 1428]
Stops a subscription for a user to a publication.

STOP SYNCHRONIZATION DELETE Statement [MobiLink] [page 1430]
Temporarily stops logging of deletes for MobiLink synchronization.

STOP SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink] [page 1432]
Stops a MobiLink synchronization schema change.

SYNCHRONIZE Statement [MobiLink] [page 1433]
Synchronizes a SQL Anywhere database with a MobiLink server. The synchronization options can be specified in the statement itself.

SYNCHRONIZE SUBSCRIPTION Statement [SQL Remote] [page 1438]
Synchronizes a subscription for a user to a publication.

SYSTEM Statement [Interactive SQL] [page 1440]
Launches an executable file from within Interactive SQL.

TRIGGER EVENT Statement [page 1441]
Triggers a named event. The event may be defined for event triggers or be a scheduled event.

TRUNCATE Statement [page 1442]
Deletes all rows from a table without deleting the table definition.

TRUNCATE TEXT INDEX Statement [page 1445]
Deletes the data in a MANUAL or an AUTO REFRESH text index.

TRY Statement [page 1446]
Implements error handling for compound statements (if an error occurs in the TRY block, it passes control to another group of statements that is enclosed in a CATCH block).

UNION Statement [page 1450]
Combines the results of two or more SELECT statements or query expressions.

UNLOAD Statement [page 1453]
Unloads data from a data source into a file.

UNPIVOT Clause [page 1459]
Unpivots compatible-type columns of a table expression in a FROM clause (FROM `unpivoted-derived-table expression`) into rows in a derived table. Unpivoting is used to normalize data, for example when you have similar data stored in multiple columns in tables and you want to return it in one column.

UPDATE Statement [page 1463]
Modifies rows in database tables.

UPDATE (Positioned) Statement [ESQL] [SP] [page 1470]
Modifies the data at the current location of a cursor.

UPDATE Statement [SQL Remote] [page 1472]
Modifies data in the database.

VALIDATE Statement [page 1476]
Validates the current database, one or more tables, materialized views, or indexes in the current database.

VALIDATE LDAP SERVER Statement [page 1479]
Validates an LDAP server configuration object.

WAITFOR Statement [page 1481]
Delays processing for the current connection for a specified amount of time or until a given time.

WAITFOR SEMAPHORE Statement [page 1483]
Decrements the counter associated with a semaphore.

WHENEVER Statement [ESQL] [page 1486]
Specifies error handling in Embedded SQL programs.

WHILE Statement [T-SQL] [page 1487]
Provides repeated execution of a statement or compound statement.

WINDOW Clause [page 1489]
Defines all or part of a window for use with window functions such as AVG and RANK in a SELECT statement.

WRITETEXT Statement [T-SQL] [page 1491]
Permits non-logged updating of a CHAR, NCHAR, or BINARY column. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

1.4.4.1 ALLOCATE DESCRIPTOR Statement [ESQL]

Allocates space for a SQL descriptor area (SQLDA).

Syntax

```
ALLOCATE DESCRIPTOR descriptor-name
[ WITH MAX { integer | hostvar } ]
```

```
descriptor-name : identifier
```

Parameters

WITH MAX clause

Allows you to specify the number of variables within the descriptor area. The default size is one. You must still call fill_sqlda to allocate space for the actual data items before doing a fetch or any statement that accesses the data within a descriptor area.

Remarks

Allocates space for a descriptor area (SQLDA). You must declare the following in your C code before using this statement:

```
struct sqlda * descriptor_name
```

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

ALLOCATE DESCRIPTOR is part of optional ANSI/ISO SQL Language Feature B031 "Basic dynamic SQL".

Example

The following sample program includes an example of ALLOCATE DESCRIPTOR statement usage.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;
#include "sqldef.h"
EXEC SQL BEGIN DECLARE SECTION;
int         x;
short       type;
int         numcols;
char        string[100];
a_SQL_statement_number  stmt = 0;
EXEC SQL END DECLARE SECTION;
int main(int argc, char * argv[]){
   struct sqlda *      sqlda1;
   if( !db_init( &sqlca ) ) {
      return 1;
   }
   db_string_connect( &sqlca,
   "UID=DBA;PWD=passwd;DBF=d:\\DB Files\\sample.db");
   EXEC SQL ALLOCATE DESCRIPTOR sqlda1 WITH MAX 25;
   EXEC SQL PREPARE :stmt FROM
      'SELECT * FROM Employees';
   EXEC SQL DECLARE curs CURSOR FOR :stmt;
   EXEC SQL OPEN curs;
   EXEC SQL DESCRIBE :stmt into sqlda1;
   EXEC SQL GET DESCRIPTOR sqlda1 :numcols=COUNT;
   // how many columns?
   if( numcols > 25 ) {
      // reallocate if necessary
      EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
      EXEC SQL ALLOCATE DESCRIPTOR sqlda1
         WITH MAX :numcols;
      EXEC SQL DESCRIBE :stmt into sqlda1;
   }
   type = DT_STRING; // change the type to string
   EXEC SQL SET DESCRIPTOR sqlda1 VALUE 2 TYPE = :type;
   fill_sqlda( sqlda1 );
   // allocate space for the variables
   EXEC SQL FETCH ABSOLUTE 1 curs
      USING DESCRIPTOR sqlda1;
   EXEC SQL GET DESCRIPTOR sqlda1
      VALUE 2 :string = DATA;
   printf("name = %s", string );
   EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
   EXEC SQL CLOSE curs;
   EXEC SQL DROP STATEMENT :stmt;
   db_string_disconnect( &sqlca, "" );
   db_fini( &sqlca );
   return 0;
}
```

Related Information

The SQL Descriptor Area (SQLDA)
DEALLOCATE DESCRIPTOR Statement [ESQL] [page 1054]

1.4.4.2 ALTER DATABASE Statement

Upgrades the database, turns jConnect support for a database on or off, calibrates the database, changes the transaction log and transaction log mirror file names, or forces a mirror server to take ownership of a database.

Syntax

Performing a system upgrade, or restoring objects

```
ALTER DATABASE UPGRADE
[ PROCEDURE ON ]
[ JCONNECT { ON | OFF } ]
[ RESTART { ON | OFF } ]
[ SYSTEM PROCEDURE AS DEFINER { ON | OFF } ]
```

Performing a user-defined upgrade

```
ALTER DATABASE UPGRADE
SCRIPT FILE sql_script_path
[ RESTART ON | OFF ]
```

Performing calibration

```
ALTER DATABASE {
 CALIBRATE [ SERVER ]
  | CALIBRATE DBSPACE dbspace-name
  | CALIBRATE DBSPACE TEMPORARY
  | CALIBRATE GROUP READ
  | CALIBRATE PARALLEL READ
  | RESTORE DEFAULT CALIBRATION
}
```

Changing transaction log and transaction log mirror names

```
ALTER DATABASE dbfile
ALTER [ TRANSACTION ] LOG
{ ON [ log-name ] [ MIRROR mirror-name ] | OFF }
[ KEY key ]
```

Changing ownership of a database

```
ALTER DATABASE
{ dbname FORCE START
| SET PARTNER FAILOVER }
```

Turning off checksum

```
ALTER DATABASE SET CHECKSUM OFF
```

Cache warming to a steady state

```
ALTER DATABASE
```

```
{ SAVE CACHE
  | RESTORE CACHE
  | DROP CACHE }
```

Parameters

dbfile

The database file. You can also specify a variable name.

PROCEDURE clause

Drop and recreate all dbo- and SYS-owned procedures in the database.

JCONNECT clause

To allow the jConnect JDBC driver access to system catalog information, specify JCONNECT ON. This clause installs the system objects that provide jConnect support. Specify JCONNECT OFF to exclude the jConnect system objects. You can still use JDBC, as long as you do not access system information. JCONNECT is ON by default.

RESTART clause

RESTART is ON by default. When RESTART ON is specified and the AutoStop (ASTOP) connection parameter is set to No, the database restarts after it is upgraded. Otherwise, the database is stopped after an upgrade.

SYSTEM PROCEDURE AS DEFINER { ON | OFF } clause

The SYSTEM PROCEDURE AS DEFINER clause specifies whether to execute pre-16.0 system procedures that perform privileged tasks with the privileges of the invoker or the definer (owner). ON means these system procedures are executed with the privileges of the definer (owner). OFF means these system procedures are executed with the privileges of the invoker.

If this clause is not specified, the default is to maintain the current behavior of the database being upgraded. When upgrading a pre-16.0 database, this means running the procedures as definer.

This setting does not impact user-defined procedures, or any procedures introduced in version 16.0 or later.

SCRIPT FILE clause

Use this clause to specify the location of the user-defined upgrade script file. `sql_script_path` is the location and name of a `.sql` script file containing the DML and DDL statements to perform.

If the script execution is not successful, and RESTART ON is specified (the default), the database is rolled back to the checkpoint that was automatically performed before the upgrade, and restarted. If RESTART OFF is specified, the database is stopped without a checkpoint, and is rolled back to the last checkpoint the next time the database is started.

CALIBRATE [SERVER] clause

Calibrate all dbspaces except for the temporary dbspace. This clause also performs the work done by CALIBRATE PARALLEL READ.

CALIBRATE DBSPACE clause

Calibrate the specified dbspace.

CALIBRATE DBSPACE TEMPORARY clause

Calibrate the temporary dbspace.

CALIBRATE GROUP READ clause

Perform group read calibration on the temporary dbspace. Writes large work tables to the temporary dbspace and uses different group read sizes to time the reading of the files. If adding space to the temporary table exceeds the limit for the connection, or if the cache is not large enough to allow calibration with the largest memory size, calibration fails and an error message is returned.

CALIBRATE PARALLEL READ clause

Calibrate the parallel I/O capabilities of devices for all dbspace files. The CALIBRATE [SERVER] clause also performs this calibration.

RESTORE DEFAULT CALIBRATION clause

Restore the Disk Transfer Time (DTT) model to the built-in default values that are based on typical hardware and configuration settings.

ALTER [TRANSACTION] LOG clause

Change the file name of the transaction log or transaction log mirror file. If MIRROR `mirror-name` is not specified, the clause sets a file name for a new transaction log. If the database is not currently using a transaction log, it starts using one. If the database is already using a transaction log, it starts using the new file as its transaction log.

If MIRROR `mirror-name` is specified, the clause sets a file name for a new transaction log mirror. If the database is not currently using a transaction log mirror, it starts using one. If the database is already using a transaction log mirror, it starts using the new file as its transaction log mirror.

You can also use this clause to turn off the transaction log or transaction log mirror. For example, ALTER DATABASE ALTER LOG OFF.

KEY clause

Specify the encryption key to use for the transaction log or transaction log mirror. The encryption key can be either a string or a variable name. When using the ALTER [TRANSACTION] LOG clause on a strongly encrypted database, you must specify the encryption key.

dbname FORCE START clause

Force a database server that is currently acting as the mirror server to take ownership of the database.

> ⚠ Caution
>
> Using the FORCE START clause can result in the loss of transactions if the primary server contains transactions that the mirror server does not have.
>
> It is recommended that you restart the primary and execute ALTER DATABASE with the SET PARTNER FAILOVER clause to force a failure without lost transactions. The FORCE START clause should only be used as a last resort when the primary cannot be restarted.

This clause can be executed from within a procedure or event, and must be executed while connected to the utility database on the mirror server.

SET PARTNER FAILOVER clause

Initiate a database mirroring failover from the primary server to the mirror server without stopping the server. This statement must be executed while you are connected to the database on the primary server, and can be executed from within a procedure or event. When this statement is executed:

1. The database server closes all connections to the database, including the connection that executed the statement
2. The database stops, and then restarts in the mirror role.
3. If the statement is contained in a procedure or event, then the other statements that follow it may not be executed

SET CHECKSUM OFF clause

Disable global checksums for the database. By default, new databases have global checksums enabled, while version 11 and earlier databases do not have global checksums enabled.

Regardless of the setting of this clause, the database server always enables write checksums for databases running on storage devices such as removable drives, to help provide early detection if the database file becomes corrupt. The database server also calculates checksums for critical pages during validation activities.

For databases that do not have global checksums enabled, you can enable write checksums by using the -wc options.

SAVE CACHE clause

Record the current cache contents so that you can restore the database to this state when necessary to improve performance.

This statement is not logged to the transaction log and cannot be executed on read-only databases, including databases involved in high availability and read-only scale-out configurations.

After the statement executes, the prefetch_pages column of the SYSDBSPACE system view is updated with one bit in the page for each page currently in cache. The bitmap that is used is the same one as that returned from the sp_db_cache_contents system procedure.

RESTORE CACHE clause

Improve performance by restoring the current database to the state it was in when the ALTER DATABASE SAVE CACHE statement was executed.

This statement is not logged to the transaction log.

This statement reads the pages identified by the prefetch_pages column of the SYSDBSPACE system view. The effect of the statement is the same as that of calling the sp_read_db_pages system procedure for each dbspace with the bitmap stored in the prefetch_column of the SYSDBSPACE system view.

DROP CACHE clause

Clear any saved cache pages.

This clause clears the pages stored in the saved_cache_pages column of the SYSDBSPACE system view and sets the column value to NULL. After executing the ALTER DATABASE DROP CACHE statement, subsequent ALTER DATABASE RESTORE CACHE statements return an error unless the ALTER DATABASE SAVE CACHE statement is executed first.

Remarks

Syntax - Upgrading components or restoring objects

Use the ALTER DATABASE UPGRADE statement as an alternative to the Upgrade utility (dbupgrad) to upgrade or update a database. By default, the database is stopped and restarted after the upgrade. The transaction log is archived during the upgrade and a new transaction log is created before the database is stopped or restarted.

After executing an ALTER DATABASE UPGRADE statement, you should shut down the database and archive the transaction log.

In general, changes in databases between minor versions are limited to additional database options and minor system table and system procedure changes. The ALTER DATABASE UPGRADE statement upgrades the system tables to the current version and adds any new database options. If necessary, it also drops and recreates all system procedures. You can force a rebuild of the system procedures by specifying the PROCEDURE ON clause.

An error message is returned if you execute an ALTER DATABASE UPGRADE statement on a database that is currently being mirrored.

You can also use the ALTER DATABASE UPGRADE statement to restore settings and system objects to their original installed state.

Features that require a physical reorganization of the database file are not made available by executing an ALTER DATABASE UPGRADE statement. Such features include index enhancements and changes in data storage. To obtain the benefits of these enhancements, you must unload and reload your database.

Back up your database files before attempting to upgrade your database.

To use the jConnect JDBC driver to access system catalog information, specify JCONNECT ON (the default). To exclude the jConnect system objects, specify JCONNECT OFF. Setting JCONNECT OFF does not remove jConnect support from a database. You can still use JDBC, as long as you do not access system catalog information. If you subsequently download a more recent version of jConnect, you can upgrade the version in the database by (re-)executing the ALTER DATABASE UPGRADE JCONNECT ON statement.

Syntax - Performing a user-defined upgrade

If `sql_script_path` is not valid, an error is returned and the database is not stopped or restarted.

The script must be stored on, and run from, the database server computer.

If the script execution is successful, the database continues running.

Syntax - Performing calibration

You can perform recalibration of the I/O cost model used by the optimizer. This operation updates the Disk Transfer Time (DTT) model, which is a mathematical model of the disk I/O used by the cost model. When you recalibrate the I/O cost model, the database server is unavailable for other use. In addition, it is essential that all other activities on the computer are idle. Recalibrating the database server is an expensive operation and may take some time to complete. Leave the default in place.

When using the CALIBRATE PARALLEL READ clause, parallel calibration is not performed on dbspace files with fewer than 10000 pages. Even though the database server automatically suspends all of its activity during calibration operations, parallel calibration should be done when there are no processes consuming significant resources on the same computer. After calibration, you can retrieve the maximum estimated number of parallel I/O operations allowed on a dbspace file by using the IOParallelism extended database property.

To eliminate repetitive, time-consuming recalibration activities when there is a large number of similar hardware installations, you can re-use a calibration by unloading it and then applying it (loading it) into

another database by using the sa_unload_cost_model and sa_load_cost_model system procedures, respectively.

Syntax - Changing transaction log and transaction log mirror names

Use the ALTER DATABASE statement to change the transaction log and transaction log mirror names associated with a database file. The database must not be running to make these changes. These changes are the same as those made by the Transaction Log (dblog) utility. You can execute this statement while connected to the utility database or another database, depending on the setting of the -gu option.

If you are changing the transaction log or transaction log mirror of an encrypted database, you must specify a key. You cannot stop using the transaction log if the database is using auditing. Once you turn off auditing, you can stop using the transaction log. This syntax is not supported in procedures, triggers, events, or batches.

Use the BACKUP DATABASE statement to rename the transaction log for a running database. For example:

```
BACKUP DATABASE DIRECTORY 'directory-name'
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

Syntax - Changing ownership of a database

ALTER DATABASE...FORCE START must be executed from the mirror server, not the primary server.

Syntax - Turning off checksum

This clause can only be used to disable checksums for a database.

Syntax - Cache warming to a steady state

Use the ALTER DATABASE statement to record the cache contents at a steady state and restore this state when necessary.

> **i** Note
>
> For parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

Unless otherwise specified in the following list, only the ALTER DATABASE system privilege is required to upgrade the database.

- **Upgrading components or restoring objects**: you must have the ALTER DATABASE system privilege, and must be the only connection to the database.
- **Performing a user-defined upgrade**: While only the ALTER DATABASE system privilege is required to execute the ALTER DATABASE statement, the user performing the upgrade must have all the privileges required to perform the actions in the specified `.sql` file. If the user does not have the proper privileges, the database upgrade fails and the database is rolled back to its state prior to executing the ALTER DATABASE statement.

- **Performing calibration**: you must have the SERVER OPERATOR system privilege, you must have file permissions on the directories where the transaction log is located, and the database must not be running. Your ability to execute the ALTER DATABASE `dbfile` ALTER TRANSACTION LOG statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.
- **Changing ownership of a database**: you must have the SERVER OPERATOR system privilege.
  The privileges required to execute an ALTER DATABASE dbname FORCE START statement can be changed by the -gd database server option.
- **Turning off checksum**: you must have the ALTER DATABASE system privilege.
- **Cache warming to a steady state**: you must have the SERVER OPERATOR system privilege.

Side Effects

Automatic commit

When executing an ALTER DATABASE UPGRADE statement, the transaction log is archived during the upgrade, and a new transaction log is created when the database restarts after the upgrade. Also, by default the database is stopped and restarted after the upgrade.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

The ALTER DATABASE statement is supported by Adaptive Server Enterprise. However, the statement's clauses supported by Adaptive Server Enterprise are disjoint from those clauses supported by SQL Anywhere.

Example

1. The following statement disables jConnect support:

   ```
   ALTER DATABASE UPGRADE JCONNECT OFF;
   ```

2. The following statement sets the transaction log file name associated with *demo.db* to `mynewdemo.log`:

   ```
   ALTER DATABASE 'demo.db\\demo.db'
      ALTER LOG ON 'mynewdemo.log';
   ```

3. The following statement performs a user-defined upgrade by running a fictitious `myUpgrade.sql` script file. If the upgrade is successful, the database continues to run. If the upgrade is not successful, the

database is rolled back to the checkpoint that was automatically performed before the upgrade, and restarted:

```
ALTER DATABASE UPGRADE SCRIPT FILE 'C:\\Users\\Public\\Documents\
\myUpgrade.sql'
   RESTART ON;
```

4. The following statement creates a variable for the database `asatest.db`:

```
CREATE VARIABLE @db1 LONG VARCHAR ;
SET @db1 = 'asatest' ;
```

   1. The following statement uses the variable @db1 to alter the database `asatest.db`:

```
ALTER DATABASE @db1
ALTER TRANSACTION LOG ON 'vis_tmp2.log'
MIRROR 'vis_tmp2.mlg' ;
```

5. The following example shows you how to save the contents of the cache when the database is running in a steady state and then restore the cache to that state when necessary.
   1. Create a dbspace by executing the following statement:

```
CREATE DBSPACE mydbs
AS 'C:\\mydb\\mydbs.db';
```

   2. Create a table in the new dbspace and add some data by executing the following statements:

```
CREATE TABLE mytable( col1 INT, col2 CHAR(128) ) IN mydbs;
```

```
INSERT INTO mytable
    SELECT column_id, column_name
    FROM sys.syscolumn;
```

```
COMMIT;
```

   3. Execute the following statement to determine the ID of the new dbspace:

```
SELECT * FROM SYS.SYSDBSPACE WHERE dbspace_name = 'mydbs';
```

      Note the ID of the new dbspace and that the saved_cache_pages column is NULL.
   4. Verify which pages from the new dbspace are currently in the cache by executing the following statement:

```
SELECT * FROM dbo.sp_db_cache_contents( );
```

      The statement above returns information for all dbspaces in the database. To restrict the results to the new mydbs dbspace, execute the following statement, where `mydbs-ID` is the ID of the new mydbs dbspace that was obtained from SYS. SYSDBSPACE:

```
SELECT * FROM dbo.sp_db_cache_contents( mydbs-ID );
```

   5. Save the current steady state of the database by executing the following statement:

```
ALTER DATABASE SAVE CACHE;
```

   6. Later, after numerous other transactions have been executed and the cache has undergone significant change, you want to return to the saved steady state. You can either read all the steady state pages for all dbspaces into the cache or read all the steady state pages for a specific dbspace into the cache.

To read all the steady state pages for all dbspaces into the cache, execute the following statement:

```
ALTER DATABASE RESTORE CACHE;
```

If you would rather read all the steady state pages for one dbspace (for example, the new dbspace mydbs), then execute a series of statements similar to the following:

```
CREATE VARIABLE cache_pages LONG VARBIT;
```

```
SELECT saved_cache_pages
    INTO cache_pages
    FROM SYS.SYSDBSPACE
    WHERE dbspace_name='mydbs';
```

```
CALL dbo.sp_read_db_pages( mydbs-ID, cache_pages );
```

Related Information

Running Pre-16.0 System Procedures as Invoker or Definer
User-defined Upgrades
Database Rebuilds
Upgrades and Rebuilds in a Database Mirroring System
Distributing Privileges Granted to the UPGRADE ROLE System Privilege After an Upgrade
Troubleshooting: The Primary Server Cannot be Restarted
CREATE DATABASE Statement [page 821]
CREATE STATISTICS Statement [page 991]
BACKUP DATABASE Statement [page 778]
Upgrade Utility (dbupgrad)
DB_EXTENDED_PROPERTY Function [System] [page 329]
-gk Database Server Option
-gu Database Server Option
-wc Database Server Option
-wc Database Option
DB_EXTENDED_PROPERTY Function [System] [page 329]

1.4.4.3 ALTER DBSPACE Statement

Preallocates space for a dbspace or for the transaction log, or updates the catalog when a dbspace file is renamed or moved.

Syntax

```
ALTER DBSPACE { dbspace-name | TRANSLOG | TEMPORARY }
  { ADD number [ add-unit ]
    | RENAME filename }
```

```
add-unit :
PAGES
| KB
| MB
| GB
| TB
```

Parameters

TRANSLOG clause

You supply the special dbspace name TRANSLOG to preallocate disk space for the transaction log. Preallocation improves performance if the transaction log is expected to grow quickly. You may want to use this feature if, for example, you are handling many binary large objects (BLOBs) such as bitmaps.

The syntax ALTER DBSPACE `dbspace-name` TRANSLOG RENAME `filename` is not supported.

TEMPORARY clause

You supply the special dbspace name TEMPORARY to add space to temporary dbspaces. When space is added to a temporary dbspace, the additional space materializes in the corresponding temporary file immediately. Preallocating space to the temporary dbspace of a database can improve performance during execution complex queries that use large work tables.

ADD clause

An ALTER DBSPACE statement with the ADD clause preallocates disk space for a dbspace. It extends the corresponding database file by the specified size, in units of pages, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). If you do not specify a unit, PAGES is the default. The page size of a database is fixed when the database is created.

If space is not preallocated, database files are extended by about 256 KB at a time for page sizes of 2 KB, 4 KB, and 8 KB, and by about 32 pages for other page sizes, when the space is needed. Pre-allocating space can improve performance for loading large amounts of data and also serves to keep the database files more contiguous within the file system.

You can use this clause to add space to any of the predefined dbspaces (system, temporary, temp, translog, and translogmirror).

RENAME clause

If you rename or move a database file other than the main file to a different directory or device, you can use ALTER DBSPACE with the RENAME clause to ensure that the database server finds the new file when the database is started. The `filename` parameter can be a string literal, or a variable.

The name change takes effect as follows:

- If the dbspace was already open before the statement was executed (that is, you have not yet renamed the actual file), it remains accessible; however, the name stored in the catalog is updated. After the database is stopped, you must rename the file to match what you provided using the RENAME clause, otherwise the file name won't match the dbspace name in the catalog and the database server is unable to open the dbspace the next time the database is started.
- If the dbspace was not open when the statement was executed, the database server attempts to open it after updating the catalog. If the dbspace can be opened, it becomes accessible. No error is returned if the dbspace cannot be opened.

To determine if a dbspace is open, execute the statement below. If the result is NULL, the dbspace is not open.

```
SELECT DB_EXTENDED_PROPERTY('FileSize','dbspace-name');
```

Using ALTER DBSPACE with RENAME on the main dbspace, system, has no effect. The RENAME clause is not supported for changing the name of the transaction log file. You can use the BACKUP DATABASE statement to rename the transaction log for a running database. For example:

```
BACKUP DATABASE DIRECTORY 'directory-name'
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

Remarks

Each database is held in one or more files. A dbspace is an additional file with a logical name associated with each database file, and used to hold more data than can be held in the main database file alone. ALTER DBSPACE modifies the main dbspace (also called the root file) or an additional dbspace. The dbspace names for a database are held in the SYSDBSPACE system view. The main database file has a dbspace name of system.

When a multi-file database is started, the start line or ODBC data source description tells the database server where to find the main database file. The main database file holds the system tables. The database server looks in these system tables to find the location of the other dbspaces, and then opens each of the other dbspaces. You can specify which dbspace new tables are created in by setting the default_dbspace option.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example increases the size of the system dbspace by 200 pages:

```
ALTER DBSPACE system
ADD 200;
```

The following example increases the size of the system dbspace by 400 MB:

```
ALTER DBSPACE system
ADD 400 MB;
```

The following example changes the file name associated with the fictitious system_2 dbspace:

```
ALTER DBSPACE system_2
RENAME 'e:\db\dbspace2.db';
```

Related Information

Predefined Dbspaces
Database File Types
CREATE DBSPACE Statement [page 830]
BACKUP DATABASE Statement [page 778]
SYSDBSPACE System View [page 1904]
default_dbspace Option

1.4.4.4 ALTER DOMAIN Statement

Renames a user-defined domain or data type.

Syntax

```
ALTER { DOMAIN | DATATYPE } user-type
RENAME new-name
```

Remarks

When you execute this statement, the name of the user-defined domain or data type is updated in the ISYSUSERTYPE system table.

> **i Note**
>
> Although domain name references in table schema are automatically updated, any procedures, triggers, views, or events that refer to the old user-defined domain or data type must be updated manually to refer to the new name.

Privileges

You must be the owner of the domain, or have one of the following privileges:

- ALTER privilege on the domain
- ALTER DATATYPE system privilege
- ALTER ANY OBJECT system privilege

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard. The ALTER DOMAIN statement is optional ANSI/ISO SQL Feature F711. However, in the standard, ALTER DOMAIN can specify modified DEFAULT or CHECK constraint clauses for an existing domain. Neither of these operations are supported in the software. Feature F711 does not support the renaming of a domain.

Example

The following example renames the fictitious Address domain to MailingAddress:

```
ALTER DOMAIN Address RENAME MailingAddress;
```

Related Information

Domains [page 192]
How to Use Domains to Improve Data Integrity
SYSUSERTYPE System View [page 1968]

CREATE DOMAIN Statement [page 836]

1.4.4.5 ALTER EVENT Statement

Changes the definition of an event or its associated handler for automating predefined actions, or alters the definition of scheduled actions. You can also use this statement to hide the definition of an event handler.

Syntax

Altering an event

```
ALTER EVENT event-name
[ AT { CONSOLIDATED | REMOTE | ALL } ]
[ FOR { PRIMARY | ALL } ]
[ { DELETE TYPE
   | TYPE event-type
   | WHERE { trigger-condition | NULL }
   | { ADD | ALTER | DELETE } SCHEDULE schedule-spec } ]
[ ENABLE | DISABLE ]
[ [ ALTER ] HANDLER compound-statement | DELETE HANDLER ]
```

```
event-type :
BackupEnd
| Connect
| ConnectFailed
| DatabaseStart
| DBDiskSpace
| Deadlock
| "Disconnect"
| GlobalAutoincrement
| GrowDB
| GrowLog
| GrowTemp
| LogDiskSpace
| RAISERROR
| ServerIdle
| TempDiskSpace
```

```
trigger-condition :
event_condition( condition-name ) { = | < | > | != | <= | >= } value |
@variable-name
```

```
schedule-spec :
[ schedule-name ]
   { START TIME start-time | BETWEEN start-time AND end-time }
   [ EVERY period { HOURS | MINUTES | SECONDS } ]
   [ ON { ( day-of-week, ... ) | ( day-of-month, ... ) } ]
   [ START DATE start-date ]
```

```
event-name | schedule-name :  identifier
```

```
day-of-week :  string
```

```
value | period | day-of-month :  integer
```

```
start-time | end-time : time

start-date : date
```

Hiding the definition of an event handler

```
ALTER EVENT event-name SET HIDDEN
```

Parameters

ALTER EVENT clause

Events do not have owners. If you specify an owner (for example, `owner.event-name`) the owner portion is ignored.

AT clause

Use this clause to change the specification regarding the databases at which the event is handled.

FOR clause

Use this clause in a database mirroring or read-only scale-out system to restrict the databases at which the event is handled.

DELETE TYPE clause

Use this clause to remove an association of the event with an event type.

ADD | ALTER | DELETE SCHEDULE clause

Use this clause to change the definition of a schedule. Only one schedule can be altered in any one ALTER EVENT statement.

WHERE clause

Use this clause to change the trigger condition under which an event is fired. The WHERE NULL option deletes a condition.

You can specify a variable name for the event_condition value.

START TIME clause

Use this clause to specify the start time and, optionally, the end time, for the event. The `start-time` and `end-time` parameters are strings (for example, `'12:34:56'`). Expressions are not allowed (for example, `NOW()`).

You can specify a variable name for `start-time`. If `start-time` is a NULL string variable, it is ignored.

BETWEEN...AND clause You can specify a variable name for `start-time` and `end-time`. If `start-time` or `end-time` are NULL string variables, they are ignored.

EVERY clause You can specify a variable name for `period`. If `period` is a NULL integer variable, the EVERY clause is ignored.

START DATE clause

Use this clause to specify the start date for the event. The `start-date` parameter is a string. Expressions are not allowed (for example, `TODAY()`). If `start-date` is a NULL string variable, it is ignored.

You can specify a variable name for `start-date`.

SET HIDDEN clause

Use this clause to hide the definition of an event handler. Specifying the SET HIDDEN clause results in the permanent obfuscation of the event handler definition stored in the action column of the ISYSEVENT system table.

Remarks

This statement allows you to alter an event definition created with CREATE EVENT. Possible uses include the following:

- hiding the definition of an event handler
- defining and testing an event handler without a trigger condition or schedule during a development phase, and then adding the conditions for execution using ALTER EVENT once the event handler is completed

Events are not owned. However, when an event is created, a user name is associated with the event (either by explicitly specifying a user name in the CREATE EVENT statement, or implicitly as the creator). The event runs with the privileges of that user name. You cannot use the ALTER EVENT statement to change the user name associated with an event (if you specify a user name, it is ignored). Instead, you must drop and recreate the event, and specify the new user name.

If you need to alter an event, you can disable it while it is running by executing an ALTER EVENT...DISABLE statement. To disable an event in *SQL Central*, right-click the event and clear the *Enabled* option. Disabling the event does not interrupt current event handler execution; the event handler continues to execute until completion. When the event handler completes, it is not restarted until you re-enable it. You can alter and then re-enable the definition. To determine what events are running, execute the following statement:

```
SELECT *
FROM dbo.sa_conn_info()
WHERE CONNECTION_PROPERTY( 'EventName',Number ) = 'event-name';
```

An `owner` specification before `event-name` is ignored.

> **i** Note
>
> For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must have either the ALTER ANY OBJECT or MANAGE ANY EVENT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

1. The following example creates then alters an event using variables for the BETWEEN clause.
   1. The following statements create three variables, one for start time, one for end time, and one for interval time:

      ```
      CREATE VARIABLE @st1 LONG VARCHAR
      CREATE VARIABLE @et1 LONG VARCHAR
      CREATE VARIABLE @int1 INTEGER
      SET @st1 = ' 8:00AM '
      SET @et1 = ' 6:00PM '
      SET @int1 = 1;
      ```

   2. The following statement creates an event that backs up the database's transaction log and uses the created variables for the BETWEEN clause:

      ```
      CREATE EVENT HourlyLogBackup
         SCHEDULE hourly_log_backup
         BETWEEN @st1 AND @et1
         EVERY @int1 HOURS ON
            ( 'Monday' , 'Tuesday' , 'Wednesday' , 'Thursday' , 'Friday' )
            HANDLER
               BEGIN
                  BACKUP DATABASE DIRECTORY 'C:\\database\\backup'
                  TRANSACTION LOG ONLY
                  TRANSACTION LOG RENAME
               END;
      ```

   3. The following statement resets the variable @st1 to 'Now':

      ```
      SET @st1 = ' Now ';
      ```

   4. The following statement alters the event by changing the start time to 'Now':

      ```
      ALTER EVENT HourlyLogBackup
            SCHEDULE hourly_log_backup
            BETWEEN @st1 AND @et1
            EVERY @int1 HOURS ON
               ( 'Monday' , 'Tuesday' , 'Wednesday' , 'Thursday' , 'Friday' )
               HANDLER
                  BEGIN
                     BACKUP DATABASE DIRECTORY 'C:\\database\backup'
                        TRANSACTION LOG ONLY
                        TRANSACTION LOG RENAME
      ```

```
            END;
```

2. The following example creates an event that runs incremental backups and then alters the event using variables for the START TIME, EVERY, and START DATE clauses.
   1. The following statement creates an event that runs an incremental backup daily at 1 AM.

```
CREATE EVENT IncrementalBackup
SCHEDULE
 START TIME '1:00 AM' EVERY 24 HOURS
   HANDLER
      BEGIN
       BACKUP DATABASE DIRECTORY 'c:\\backup'
       TRANSACTION LOG ONLY
       TRANSACTION LOG RENAME MATCH
      END;
```

3. The following statements create three variables, one for start time, one for start date, and one for interval time:

```
CREATE VARIABLE @st2 LONG VARCHAR
CREATE VARIABLE @sd2 LONG VARCHAR
CREATE VARIABLE @int2 INTEGER
SET @st2 = ' 3:00AM '
SET @sd2 = '2013-01-01'
SET @int2 = 24;
```

4. The following statement alters the event IncrementalBackup and uses variables in the START TIME, EVERY, and START DATE clauses:

```
ALTER EVENT IncrementalBackup
   SCHEDULE
      START TIME @st2 EVERY @int2 HOURS
      START DATE @sd2
         HANDLER
            BEGIN
               BACKUP DATABASE DIRECTORY 'c:\\backup'
               TRANSACTION LOG ONLY
               TRANSACTION LOG RENAME MATCH
            END;
```

Related Information

System Events
Hiding an Event Handler
SYSEVENT System View [page 1906]
BEGIN Statement [page 784]
CREATE EVENT Statement [page 847]

1.4.4.6 ALTER EXTERNAL ENVIRONMENT Statement

Specifies the location of an external environment such as Java, PHP, or Perl.

Syntax

```
ALTER EXTERNAL ENVIRONMENT environment-name
LOCATION location-string
```

```
environment-name :
C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64
| CLR
| DBMLSYNC
| JAVA
| JS
| PERL
| PHP
```

Parameters

environment-name

Use `environment-name` to specify the external environment you are altering.

location-string

Use `location-string` to specify the location on the database server computer where the executable/binary for the external environment can be found. It includes the executable/binary name. This path can either be fully qualified or relative. If the path is relative, then the executable/binary must be in a location where the server can find it.

Remarks

In general, the ALTER EXTERNAL ENVIRONMENT statement is used to help the database server locate an external environment module when it cannot be located using standard search techniques (such as searching the software install location, system PATH, etc.).

CLR

The LOCATION string is used to identify both the location and version of the CLR external environment support module. For example, `dbextclr[VER_MAJOR]_v4.5` indicates that support for .NET 4.x is required. In this example, the file path is not specified so the database server will locate the module using standard search techniques. The `dbextclr[VER_MAJOR]` module supports .NET 2 and 3.5. The `dbextclr[VER_MAJOR]_v4.5` module supports .NET 4.x. For portability to newer versions of the software, use *[VER_MAJOR]* in the LOCATION string rather than the current software release version number. The database server will replace [VER_MAJOR] with the appropriate version number.

If the CLR external environment cannot be started, make sure the corresponding Sap.Data.SQLAnywhere assembly is installed using SetupVSPackage. For example, `Sap.Data.SQLAnywhere.v4.5.dll` must be installed when the following statement has been executed.

```
ALTER EXTERNAL ENVIRONMENT CLR LOCATION 'dbextclr[VER_MAJOR]_v4.5';
```

DBMLSYNC

In cases where there are multiple versions of the database server software installed on the same system, or if the dbmlsync executable is not located in the same directory as the database server and cannot be found in the PATH environment variable, use the ALTER EXTERNAL ENVIRONMENT command to specify the location of the dbmlsync executable.

Privileges

You must have the MANAGE ANY EXTERNAL ENVIRONMENT system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example specifies the location of the Perl executable for use when using Perl as an external environment.

```
ALTER EXTERNAL ENVIRONMENT PERL
LOCATION 'c:\\Perl64\\bin\\perl.exe';
```

Related Information

External Environment Support
START EXTERNAL ENVIRONMENT Statement [page 1410]

STOP EXTERNAL ENVIRONMENT Statement [page 1423]
INSTALL EXTERNAL OBJECT Statement [page 1238]
REMOVE EXTERNAL OBJECT Statement [page 1333]
SYSEXTERNENV System View [page 1908]
SYSEXTERNENVOBJECT System View [page 1910]

1.4.4.7 ALTER FUNCTION Statement

Modifies a function.

Syntax

Change the definition of a function

```
ALTER FUNCTION [ owner.]function-name function-definition
```

```
function-definition : see the CREATE FUNCTION statement
```

Obfuscate a function definition

```
ALTER FUNCTION [ owner.]function-name
SET HIDDEN
```

Recompile a function

```
ALTER FUNCTION [ owner.]function-name
RECOMPILE
```

Remarks

You must include the entire new function in the ALTER FUNCTION statement.

Change the definition of a function

The ALTER FUNCTION statement is identical in syntax to the CREATE FUNCTION statement except for the first word.

With ALTER FUNCTION, existing privileges on the function remain unmodified. Conversely, if you execute DROP FUNCTION followed by CREATE FUNCTION, execute privileges are reassigned.

Obfuscate a function definition

Use SET HIDDEN to obfuscate the definition of the associated function and cause it to become unreadable. The function can be unloaded and reloaded into other databases.

If SET HIDDEN is used, then debugging using the debugger does not show the function definition, nor is it available through procedure profiling.

> **Note**
>
> *This setting is irreversible.* Retain the original function definition outside of the database.

Recompile a function

Use the RECOMPILE syntax to recompile a user-defined SQL function. When you recompile a function, the definition stored in the catalog is re-parsed and the syntax is verified. The preserved source for a function is not changed by recompiling. When you recompile a function, the definitions obfuscated by the SET HIDDEN clause remain obfuscated and unreadable.

> **i** Note
>
> For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must be the owner of the function or have one of the following privileges:

- ALTER ANY PROCEDURE system privilege
- ALTER ANY OBJECT system privilege

To make a function external, you must have the CREATE EXTERNAL REFERENCE system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

ALTER FUNCTION is optional ANSI/ISO SQL Language Feature F381. However, in the SQL standard, ALTER FUNCTION cannot be used to re-define a SQL Persistent Stored Module (PSM) function definition. The ANSI/ISO SQL Standard does not include support for SET HIDDEN or RECOMPILE.

Example

1. In this example, MyFunction is created and altered. The SET HIDDEN clause obfuscates the function definition and makes it unreadable. To run this example, you must also have the CREATE PROCEDURE system privilege, since a function is being created before being altered.

```
CREATE FUNCTION MyFunction(
   firstname CHAR(30),
   lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
   DECLARE name CHAR(61);
   SET name = firstname || ' ' || lastname;
   RETURN (name);
ALTER FUNCTION MyFunction SET HIDDEN;
END;
```

2. The following example creates and then alters a function using a variable in the NAMESPACE clause
   1. The following statements create a variable for a NAMESPACE clause:

```
CREATE VARIABLE @ns LONG VARCHAR ;
SET @ns = 'http://wsdl.domain.com/' ;
```

   2. The following statement creates a function named FtoC that uses a variable in the NAMESPACE clause:

```
CREATE FUNCTION FtoC ( IN temperature LONG VARCHAR )
RETURNS LONG VARCHAR
URL 'http://localhost:8082/FtoCService'
TYPE 'SOAP:DOC'
NAMESPACE @ns;
```

   3. The following statement alters the function FtoC so that it accepts and returns a FLOAT data type:

```
ALTER FUNCTION FtoC ( IN temperature FLOAT )
RETURNS FLOAT
URL 'http://localhost:8082/FtoCService'
NAMESPACE @ns;
```

Related Information

Hiding the Contents of a Procedure, Function, Trigger, Event, or View
CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [External Call] [page 866]
CREATE FUNCTION Statement [Web Service] [page 874]
ALTER PROCEDURE Statement [page 704]
DROP FUNCTION Statement [page 1097]

1.4.4.8 ALTER INDEX Statement

Renames an index, primary key, or foreign key, or changes the clustered nature of an index.

Syntax

```
ALTER { INDEX index-name
| [ INDEX ] FOREIGN KEY role-name
| [ INDEX ] PRIMARY KEY }
ON [ owner.]object-name { REBUILD | rename-clause | cluster-clause }
```

```
object-name : table-name | materialized-view-name
```

```
rename-clause : RENAME { AS | TO } new-index-name
```

```
cluster-clause : CLUSTERED | NONCLUSTERED
```

Parameters

rename-clause

Specify the new name for the index, primary key, or foreign key.

When you rename the underlying index for a foreign or primary key, the corresponding RI constraint name for the index is not changed. However, the foreign key role name, if applicable, is the same as the index name and is changed. Use the ALTER TABLE statement to rename the RI constraint name, if necessary.

cluster-clause

Specify whether the index should be changed to CLUSTERED or NONCLUSTERED. Only one index on a table can be clustered.

REBUILD clause

Use this clause to rebuild an index, instead of dropping and recreating it.

Remarks

The ALTER INDEX statement carries out two tasks:

- It can be used to rename an index, primary key, or foreign key.
- It can be used to change an index type from nonclustered to clustered, or vice versa.
  The ALTER INDEX statement can be used to change the clustering specification of the index, but does not reorganize the data. As well, only one index per table or materialized view can be clustered.

ALTER INDEX cannot be used to change an index on a local temporary table. An attempt to do so results in an Index not found error.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

To alter an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

To alter an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

Side Effects

Automatic commit. Closes all cursors for the current connection. If ALTER INDEX REBUILD is specified, a checkpoint is performed.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement changes IX_product_name to be a clustered index:

```
ALTER INDEX IX_product_name ON GROUPO.Products
CLUSTERED;
```

The following statement renames the index IX_product_name on the Products table to ixProductName:

```
ALTER INDEX IX_product_name ON GROUPO.Products
RENAME TO ixProductName;
```

Related Information

Snapshot Isolation
CREATE INDEX Statement [page 886]
ALTER TABLE Statement [page 742]

1.4.4.9 ALTER LDAP SERVER Statement

Alters an LDAP server configuration object.

Syntax

```
ALTER LDAP SERVER ldapua-server-name
[ ldapua-server-attribs ... ]
[ WITH { SUSPEND | ACTIVATE | REFRESH } ]
```

```
ldapua-server-attribs :
SEARCH DN search-dn-attributes ...
| AUTHENTICATION URL { 'url-string' | NULL }
| CONNECTION TIMEOUT timeout-value
| CONNECTION RETRIES retry-value
| TLS { ON | OFF }
```

```
search-dn-attributes :
URL { 'url-string' | NULL }
| ACCESS ACCOUNT { 'dn-string' | NULL }
| IDENTIFIED BY ( 'password' | NULL }
| IDENTIFIED BY ENCRYPTED { encrypted-password | NULL }
```

Parameters

SEARCH DN clause

There is no default value for any parameter in the SEARCH DN clause.

URL

Use this clause to specify the host (by name or by IP address), port number, and search to be performed to do the lookup of the **LDAP Distinguished Name (DN)** for a given user ID. `url-string` is validated for correct LDAP URL syntax before it is stored in ISYSLDAPSERVER. The maximum size for this string is 1024 bytes.

The format of `url-string` must comply with the LDAP URL standard. See The LDAP Standard Specification .

ACCESS ACCOUNT

Use this clause to specify the LDAP Distinguished Name (DN) used by the database server to connect to the LDAP server. This is not a SQL Anywhere user, but a user created in the LDAP server specifically for logging in to the LDAP server. This user must have permissions within the LDAP server to search for DNs by user ID in the locations specified in the SEARCH DN URL clause. The maximum size for this string is 1024 bytes.

IDENTIFIED BY

Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT. The maximum size is 255 bytes, and cannot be set to NULL.

IDENTIFIED BY ENCRYPTED

Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT, provided in encrypted form, and is a binary value stored somewhere on disk. The maximum size of the

binary is 289 bytes, and cannot be set to NULL. IDENTIFIED BY ENCRYPTED allows the password to be retrieved and used, without it becoming known.

AUTHENTICATION URL clause

Use this clause to specify the host by name or IP address, and the port number of the LDAP server to use to authenticate a user. The DN of the user obtained from a prior DN search and the user password are used to bind a new connection to the authentication URL. A successful connection to the LDAP server is considered proof of the identity of the connecting user. There is no default value for this parameter.

CONNECTION TIMEOUT clause

Use this clause to specify the connection timeout, in milliseconds, to the LDAP server, both for searches for the DN and for authentication. The default value is 10 seconds.

CONNECTION RETRIES clause

Use this clause to specify the number of retries for connections to the LDAP server, both for searches for the DN and for authentication. The valid range of values is 1-60. The default is 3.

TLS clause

Use this clause to specify the use of the TLS protocol on connections to the LDAP server, both for the DN searches, and for authentication. The valid values are ON or OFF. The default is OFF. Use the Secure LDAP protocol by specifying `ldaps://` to begin the URL instead of `ldap://`. The TLS option must be set to OFF when using Secure LDAP.

WITH clause

WITH SUSPEND

Sets the state of the LDAP server communications to SUSPENDED (maintenance mode). The connections to the LDAP server are closed and authentication with the LDAP server is no longer performed.

WITH ACTIVATE

Activates the LDAP server for immediate use. This changes the state of the LDAP server communications to READY.

WITH REFRESH

Reinitializes LDAP user-authentication. This command does not change the state of the LDAP server if it is in the SUSPENDED state. When WITH REFRESH is specified for an LDAP server in the READY or ACTIVE state, connections to the LDAP server are closed. Then, the server option values are reread from the ISYSLDAPSERVER system table and are applied to new connections to the LDAP server and to incoming authentication requests to the database server.

Remarks

ALTER LDAP SERVER...WITH REFRESH is often used on an LDAP server that is in the ACTIVE or READY state to release any resources that may be held, or to reread changes made to files outside of the server, such as a change to the contents of the file specified by the trusted_certificates_file database option.

For other states, ALTER LDAP SERVER...WITH REFRESH has no effect.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example suspends the fictitious LDAP server named apps_primary.

```
ALTER LDAP SERVER apps_primary WITH SUSPEND;
```

The following example changes the LDAP server named apps_primary to use a different URL for authentication on host fairfax, port number 1066, sets connection retries to 10, and activates it.

```
ALTER LDAP SERVER apps_primary
 AUTHENTICATION URL 'ldap://fairfax:1066/'
 CONNECTION RETRIES 10
 WITH ACTIVATE;
```

Related Information

SYSLDAPSERVER System View [page 1920]
CREATE LDAP SERVER Statement [page 891]
DROP LDAP SERVER Statement [page 1100]
VALIDATE LDAP SERVER Statement [page 1479]

1.4.4.10 ALTER LOGIN POLICY Statement

Alters an existing login policy.

Syntax

```
ALTER LOGIN POLICY policy-name policy-options

policy options :
policy-option [ policy-option ... ]

policy-option :
policy-option-name = policy-option-value

policy-option-value :
{ UNLIMITED
| DEFAULT
| legal-option-value }
```

Parameters

policy-name

The name of the login policy. Specify root to modify the root login policy.

policy-option-name

The name of the policy option.

policy-option-value

The value assigned to the login policy option. If you specify UNLIMITED, no limits are used. If you specify DEFAULT, the default limits are used.

Remarks

When a login policy is altered, changes are immediately applied to all users.

If you do not specify a policy option, values for this login policy are taken from the root login policy. New policies do not inherit the MAX_NON_DBA_CONNECTIONS and ROOT_AUTO_UNLOCK_TIME policy options.

All new databases include a root login policy. You can modify the root login policy values, but you cannot delete the policy. An overview of the default values for the root login policy is provided in the table above.

Privileges

You must have the MANAGE ANY LOGIN POLICY system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example alters the fictitious Test1 login policy by changing the LOCKED and MAX_CONNECTIONS policy options. The LOCKED value indicates that users with the policy cannot establish new connections and the MAX_CONNECTIONS value limits the number of concurrent connections that are allowed.

```
ALTER LOGIN POLICY Test1
LOCKED=ON
MAX_CONNECTIONS=5;
```

This example overrides the root login policy LOCKED and MAX_CONNECTIONS policy options.

```
ALTER LOGIN POLICY root
LOCKED=ON
MAX_CONNECTIONS=5;
```

The following example sets a primary and a secondary LDAP server for a fictitious ldap_user_policy login policy, and turns off the ability to failover to standard authentication, even when database option login_mode includes 'Standard'. This provides strict controls on users of this login policy so that only LDAP user authentication is used for authentication. In the event that a high volume of login connections occur such that the LDAP server is unable to respond and authenticate quickly, users whose retries and timeouts are exhausted will see connection failures to the database server rather than failover to use standard authentication.

```
ALTER LOGIN POLICY ldap_user_policy
LDAP_PRIMARY_SERVER=ldapsrv1
LDAP_SECONDARY_SERVER=ldapsrv2
LDAP_FAILOVER_TO_STD=OFF;
```

The following example resets the timestamp value for a fictitious application_user_policy login policy to the current time. Any user that is assigned this policy have their Distinguished Name (DN) searched on the next login attempt, rather than using the value cached in ISYSUSER. This strategy purges old DN values held in ISYSUSER for users associated with this policy at the time of their next authentication.

```
ALTER LOGIN POLICY application_user_policy
LDAP_REFRESH_DN=NOW;
```

Related Information

Login Policies
Altering a Login Policy
ALTER USER Statement [page 769]
COMMENT Statement [page 808]
CREATE LOGIN POLICY Statement [page 894]
CREATE USER Statement [page 1045]
DROP LOGIN POLICY Statement [page 1101]
DROP USER Statement [page 1143]
Login Policy Options and Default Values

1.4.4.11 ALTER MATERIALIZED VIEW Statement

Alters a materialized view.

Syntax

```
ALTER MATERIALIZED VIEW [ owner.]materialized-view-name {
 SET HIDDEN
| { ENABLE | DISABLE }
| { ENABLE | DISABLE } USE IN OPTIMIZATION
| { ADD PCTFREE percent-free-space | DROP PCTFREE }
| [ NOT ] ENCRYPTED
| [ { IMMEDIATE | MANUAL } REFRESH ]
}
```

```
percent-free-space : integer
```

Parameters

SET HIDDEN clause

Use the SET HIDDEN clause to obfuscate the definition of a materialized view. *This setting is irreversible*.

ENABLE clause

Use the ENABLE clause to enable a disabled materialized view, making it available for the database server to use. This clause has no effect on a view that is already enabled. After using this clause, you must refresh the view to initialize it, and recreate any text indexes that were dropped when the view was disabled.

DISABLE clause

Use the DISABLE clause to disable use of the view by the database server. When you disable a materialized view, the database server drops the data and indexes for the view.

{ ENABLE | DISABLE } USE IN OPTIMIZATION clause

Use this clause to specify whether you want the materialized view to be available for the optimizer to use. If you specify DISABLE USE IN OPTIMIZATION, the materialized view is used only when executing queries that explicitly reference the view. The default is ENABLE USE IN OPTIMIZATION.

ADD PCTFREE clause

Specify the percentage of free space you want to reserve on each page. The free space is used if rows increase in size when the data is updated. If there is no free space on a page, every increase in the size of a row on that page requires the row to be split across multiple pages, causing row fragmentation and possible performance degradation.

The value of `percent-free-space` is an integer between 0 and 100. The value 0 specifies that no free space is to be left on each page. Each page is to be fully packed. A high value causes each row to be inserted into a page by itself. If PCTFREE is not set, or is dropped, the default PCTFREE setting is applied according to the database page size (200 bytes for a 4 KB page size, and 100 bytes for a 2 KB page size).

DROP PCTFREE clause

Removes the PCTFREE setting currently in effect for the materialized view, and applies the default PCTFREE according to the database page size.

[NOT] ENCRYPTED clause

Specify whether to encrypt the materialized view data. By default, materialized view data is not encrypted at creation time. To encrypt a materialized view, specify ENCRYPTED. To decrypt a materialized view, specify NOT ENCRYPTED.

REFRESH clause

Use the REFRESH clause to change the refresh type for the materialized view:

IMMEDIATE REFRESH

Use the IMMEDIATE REFRESH clause to change a manual view to an immediate view. The manual view must be valid and uninitialized to change the refresh type to IMMEDIATE REFRESH. If the view is in an initialized state, execute a TRUNCATE statement to change the state to uninitialized before executing the ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH.

MANUAL REFRESH

Use the MANUAL REFRESH clause to change an immediate view to a manual view.

Remarks

If you alter a materialized view owned by another user, you must qualify the name by including the owner (for example, GROUPO.EmployeeConfidential). If you don't qualify the name, the database server looks for a materialized view with that name owned by you and alters it. If there isn't one, it returns an error.

When you disable a materialized view (DISABLE clause), it is no longer available for the database server to use for answering queries. As well, the data and indexes are dropped, and the refresh type changes to manual. Any dependent regular views are also disabled.

The DISABLE clause requires exclusive access not only to the view being disabled, but to any dependent views, since they are also disabled.

Table encryption must already be enabled on the database to encrypt a materialized view (ENCRYPTED clause). The materialized view is then encrypted using the encryption key and algorithm specified at database creation time.

The only operations a user can perform on a materialized view to change its data are refreshing, truncating, and disabling. However, immediate views are automatically updated by the database server. That is, once an immediate view is enabled and initialized, the database server maintains it automatically, without additional privileges checking.

Privileges

You must be the owner of the materialized view, or have the ALTER ANY MATERIALIZED VIEW or ALTER ANY OBJECT system privilege.

If you do not have a required privilege but want to alter a materialized view to be immediate (ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH), you must own the view and all the tables it references.

Side Effects

- Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements creates the EmployeeConfid88 materialized view and then disables its use in optimization. To run this example you must also have the CREATE ANY MATERIALIZED VIEW system privilege, as well as SELECT privilege on the Employees and Departments tables.

> ⚠ Caution
>
> When you are done with this example, you should drop the materialized view you created. Otherwise, you cannot make schema changes to its underlying tables, Employees and Departments, when trying out other examples. You cannot alter the schema of a table that has enabled, dependent materialized view.

```
CREATE MATERIALIZED VIEW EmployeeConfid88 AS
   SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary,
ManagerID,
      Departments.DepartmentName, Departments.DepartmentHeadID
   FROM GROUPO.Employees, GROUPO.Departments
   WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid88;
ALTER MATERIALIZED VIEW EmployeeConfid88 DISABLE USE IN OPTIMIZATION;
```

Related Information

Materialized Views
View Dependencies
Materialized Views Restrictions
Whether to Set Refresh Type to Manual or Immediate
Advanced: Status and Properties for Materialized Views
Enabling or Disabling a Materialized View
Encrypting or Decrypting a Materialized View
Hiding a Materialized View Definition
Enabling or Disabling Optimizer Use of a Materialized View
CREATE MATERIALIZED VIEW Statement [page 896]
REFRESH MATERIALIZED VIEW Statement [page 1321]
DROP MATERIALIZED VIEW Statement [page 1103]
TRUNCATE Statement [page 1442]
sa_refresh_materialized_views System Procedure [page 1681]

1.4.4.12 ALTER MIRROR SERVER Statement

Modifies the attributes of a mirror server.

Syntax

Alter a mirror server

```
ALTER MIRROR SERVER mirror-server-name
[AS { PRIMARY | MIRROR | ARBITER | PARTNER}]
[ server-option = { string | NULL } [ ... ] ]
```

Alter a mirror server as a copy

```
ALTER MIRROR SERVER mirror-server-name
[AS COPY]
[ { FROM SERVER parent-name [ OR SERVER server-name ] | USING AUTO
PARENT } | ALTER PARENT FROM mirror-server-name ]
[ server-option = { string | NULL } [ ... ] ]
```

```
parent-name :
server-name | PRIMARY
```

```
server-option :
connection_string
logfile
preferred
state_file
```

Parameters

AS clause

Use the AS clause to change the server-type of the mirror server from PARTNER to COPY or COPY to PARTNER. This clause is not needed and it is not recommended if you are not changing the type of mirror server.

PARTNER

Only a database server with the mirror server type of COPY can use this value to change its type to PARTNER. The parent definitions for the copy node are deleted.

The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

In a database mirroring system, the partners use the connection string value to connect to each other. In a read-only scale-out system, the connection string is used by a copy-node that has this server as its parent.

COPY

Only a database server whose mirror server type is PARTNER can use this value to change its server type to a copy node. This partner server must also currently have the MIRROR role.

In a read-only scale-out system, this value specifies that the database server is a copy node. All connections to the database on this server are read-only. The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

FROM SERVER clause

This clause can only be used when AS COPY is specified. This clause constructs a tree of servers for a scale-out system and indicates which servers the copy nodes obtain transaction log pages from.

The parent can be specified using the name of the mirror server or PRIMARY. An alternate parent for the copy node can be specified using the OR SERVER clause.

In a database mirroring system that has only two levels (partner and copy nodes), the copy nodes obtain transaction log pages from the current primary or mirror server.

A copy node determines which server to connect to by using its mirror server definition that is stored in the database. From its definition, it can locate the definition of its parent, and from its parent's definition, it can obtain the connection string to connect to the parent.

You do not have to explicitly define copy nodes for the scale-out system: you can choose to have the root node define the copy nodes when they connect.

USING AUTO PARENT clause

This clause can only be used when AS COPY is specified. This clause causes the primary server to assign a parent for this server.

ALTER PARENT FROM clause

This clause can only be used when AS COPY is specified. This clause changes the parent for this mirror server, and assigns all its siblings to be its children. The server name specified by the ALTER PARENT FROM clause is used to verify that the current parent for this server matches the value specified. This is

used to ensure that only one of a collection of siblings is able to replace its parent if they all request the change simultaneously.

server-option clause

You can specify a variable name for `server-option`.

The following options are also supported:

connection_string server option

Specifies the connection string to be used to connect to the server. The connection string for a mirror server should not include a user ID or password because they are not used when one mirror server connects to another mirror server.

logfile server option

Specifies the location of the file that contains one line per request that is sent between mirror servers if database mirroring is used. This file is used only for debugging.

preferred server option

Specifies whether the server is the preferred server in the mirroring system. You can specify either YES or NO. The preferred server assumes the role of primary server whenever possible. You specify this option when defining PARTNER servers.

state_file server option

Specifies the location of the file used for maintaining state information about the mirroring system. This option is required for database mirroring. In a mirroring system, a state file must be specified for servers with type PARTNER. For arbiter servers, the location is specified as part of the command to start the server.

Remarks

Read-only scale-out and database mirroring each require a separate license.

In a database mirroring system, the mirror server type can be PRIMARY, MIRROR, ARBITER, or PARTNER.

In a read-only scale-out system, the mirror server type can be PRIMARY, PARTNER, or COPY.

You can only change the mirror server type from COPY to PARTNER or from PARTNER to COPY. To change to or from a PRIMARY, MIRROR, or ARBITER server type, you must drop the mirror server definition and recreate it.

Mirror server names for servers of type PARTNER and COPY must match the names of the database servers that will be part of the mirroring system (the name used with the -n server option). This requirement allows each database server to find its own definition and that of its parent. `mirror-server-name`, `parent-name`, and `server-name` above must contain only 7-bit ASCII characters.

When you convert from a copy node to a partner, the parent definitions are deleted from the mirror server definition.

To replace a mirror server definition, use the CREATE MIRROR SERVER statement with the OR REPLACE clause.

> **Note**
>
> For parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must have the MANAGE ANY MIRROR SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

1. The following example does the following:
   1. Creates a primary server called scaleout_primary1.
   2. Creates a copy node, scaleout_child1, for scaleout_primary1.
   3. Creates a mirror server, scaleout_mirror1.
   4. Using the ALTER MIRROR SERVER statement, reassigns scaleout_child1 to be a copy node of scaleout_mirror1.

```
CREATE MIRROR SERVER "scaleout_primary1"
   AS PRIMARY
   connection_string =
'server=scaleout_primary1;host=winxp-2:6871,winxp-3:6872';
CREATE MIRROR SERVER "scaleout_child1"
   AS COPY FROM SERVER  "scaleout_primary1"
   connection_string = 'server=scaleout_child1;host=winxp-2:6878';
CREATE MIRROR SERVER "scaleout_mirror1"
   AS MIRROR
   connection_string =
'server=scaleout_mirror1;host=winxp-2:6871,winxp-3:6872';
```

```
ALTER MIRROR SERVER "scaleout_child1"
   FROM SERVER "scaleout_mirror1"
   connection_string = 'server=scaleout_child1;host=winxp-2:6878';
```

2. The following example alters a mirror server using a variable for the `connection_string` parameter.
   1. The following statement creates a variable for a connection string:

```
CREATE VARIABLE @connstr_value LONG VARCHAR
SET @connstr_value = ' server=new_scaleout_primary;host=winxp-2:6878 ';
```

   2. The following statement alters the mirror server new_scaleout_primary using the variable @connstr_value for the `connection_string` parameter:

```
ALTER MIRROR SERVER new_scaleout_primary AS PRIMARY
connection_string = @connstr_value = @connstr_value;
```

Related Information

How Child Copy Nodes Are Added
Alphabetical List of Connection Parameters
Troubleshooting: State Information Files of the Partners and Arbiter
Automatically Assign the Parent of a Copy Node
Preferred Database Server in a Database Mirroring System
Separately Licensed Components
Converting a Partner Server to a Copy Node
SYSMIRRORSERVER System View [page 1924]
CREATE MIRROR SERVER Statement [page 901]
COMMENT Statement [page 808]
DROP MIRROR SERVER Statement [page 1105]

1.4.4.13 ALTER ODATA PRODUCER Statement

Alters an OData Producer.

Syntax

```
ALTER ODATA PRODUCER name [ producer-clause... ]
```

```
producer-clause:
[ ADMIN USER { NULL | user } ]
[ AUTHENTICATION [ DATABASE | USER user ] ]
[ [ NOT ] ENABLED ]
[ MODEL [ FROM ] { FILE string-or-variable [ ENCODING string ] | VALUE
string-or-variable [ ENCODING string ] } ]
[ [ SERVICE ] ROOT string ]
[ USING { string | NULL } ]
```

Parameters

ADMIN USER clause Specifies an administrative user that the OData Producer uses to create tables, a procedure, and an event to manage repeatable requests. The user must have the CREATE TABLE, CREATE PROCEDURE, MANAGE ANY EVENT, and VERIFY ODATA system privileges. This user cannot be the same user specified by the AUTHENTICATION clause.

AUTHENTICATION clause Specifies how the OData server connects to the database. Specify the DATABASE for users to connect with personalized credentials. These credentials are requested by using Basic HTTP authentication. Specify the USER for all users to connect by using the connection string of the specified user. The specified user must have the VERIFY ODATA system privilege. The USER is recommended for testing and read-only producers.

ENABLED clause Specifies whether this OData Producer is enabled or disabled. By default, OData Producers are enabled.

MODEL clause

Specifies the OData Producer service model (given in OSDL) that indicates which tables and views are exposed in the OData metadata. If you specify the FILE clause, then the value must be a variable or string of the file name that contains the OSDL data. If the file name is not fully qualified, then the path is relative to the database server's current working directory. If the file name is specified, then the file is read and the contents of the file are stored in the database and logged in the transaction log. By default, tables and views are exposed based on user privileges. Tables and views without primary keys are not exposed.

Use the ENCODING clause to specify the character set of the model data or file. The default is UTF-8.

SERVICE ROOT clause

Specifies the root of the OData service on the OData server. All resources for the OData Producer are accessed by using URIs of the following format: `scheme:host:port/path-prefix/resource-path[query-options]`.

The SERVICE ROOT clause is `/path-prefix`. The value must be specified, it cannot be NULL, and it must start with a **/**. A `path-prefix` must be a validly encoded URI path component (for example, spaces escaped as %20, % escaped as %25, and non US-ASCII characters encoded UTF-8 and escaped). Service roots cannot include the characters `:?[']#@` (neither encoded nor unencoded), and they cannot include the encoded version of the / character and should not include . and .. path segments.

USING clause

This string specifies additional OData Producer options. Options are specified in a semicolon-separated list of name=value pairs. If the USING clause is altered, then you must repeat all values that you want to retain.

Option	Description
AccessControlAllowOrigins { `values` | * | *none*	This parameter allows you to configure the OData Producer to return the correct headers with Cross-origin Resource Sharing (CORS) requests. The producer responds to Origin and Access-Control-Request-Method HTTP request headers when the AccessControlAllowedOrigins configuration parameter is specified. The AccessControlAllowedMethods configuration parameter can be used to limit what HTTP methods are allowed to respond to CORS requests. `values` is a comma-separated list of origins that are allowed to access the resources. * means all origins. *none* (the default) means do not enable for this producer. If an allowed origin contains one or more * characters (for example http://*.domain.com), then * characters are converted to ".*". Any . characters are escaped to "\." and the resulting allowed origin interpreted as a regular expression. Allowed origins can be more complex expressions such as https?://*.domain.[a-z]{3} that match HTTP or HTTPS, multiple subdomains, and any three-letter top-level domain (.com, .net, .org, and so on.). https?://mydomain.com is not treated as a pattern because it contains no * characters. Use http://mydomain.com,https://mydomain.com instead. This option cannot be set to none if you are using AccessControlAllowMethods.
AccessControlAllowMethods `values`	Specifies the allowed methods for Cross-Origin Resource Sharing `values` is a comma-separated list of HTTP methods that are allowed when accessing the resources. The default value is GET,POST,HEAD. Allowed values are GET, HEAD, POST, PUT, PATCH, MERGE, and DELETE. To use AccessControlAllowMethods, you must specify a value for AccessControlAllowOrigins.
ConnectionAuthExpiry = `num-seconds`	Specifies how long heavily used pooled connections are valid for. The default time is 5 minutes, the minimum time is 1 second, and the maximum time is 24 hours.

Option	Description
ConnectionPoolMaximum = `num-max-connections`	Indicates the maximum number of simultaneous connections that this OData Producer keeps open for use in the connection pool. Fewer connections might be used by the connection pool depending on the OData server load. By default, the connection pool size is limited to half of the maximum number of simultaneous connections that the database server supports.
CSRFTokenTimeout = `num-seconds-valid`	Enables CSRF token checking and specifies the number of seconds that a token is valid for. By default, this value is 0, which disables CSRF token checking. Otherwise, the number of seconds must be a valid integer value from 1 to 1800.
PageSize = `num-max-entities`	Specifies the maximum number of entities to include in a retrieve entity set response before issuing a next link. The default setting is 100.
ReadOnly = { *true* | *false* }	Indicates whether modification requests should be ignored. The default setting is false.
RepeatRequestForDays = { `days-number` }	Specifies how long repeatable requests are valid. This value must be an integer ranging from 1 to 31. The default setting is 2. This option is only effective when the ADMIN USER clause is not NULL.
SecureOnly = { *true* | *false* }	Indicates whether the Producer should only listen for requests on the HTTPS port. The default setting is false.
ServiceOperationColumnNames = { *generate* | *database* }	Specifies whether the column names in the metadata should be generated or taken from the result set columns in the database when naming the properties of the ComplexType used in the ReturnType. The default setting is generate.

Remarks

Alters an OData Producer that runs as a sub-process of the database server.

Privileges

You must have the MANAGE ODATA system privilege.

If you specify the AUTHENTICATION USER clause or the ADMIN USER clause, then the specified users must have the VERIFY ODATA system privilege.

Side Effects

If the OData Producer is running, then it may be stopped and restarted.

Example

Assume that you want to create an OData Producer that accesses objects owned by user Dave through a URL beginning with `/odata/dave` and that you have a model file called `dave.txt` that contains details of the tables that you want to expose. You also want to ensure that no data can be updated through the OData interface and that requests are repeated for up to ten days. Create an OData Producer called dave_producer by executing the following statement:

```
CREATE ODATA PRODUCER dave_producer
MODEL FROM FILE 'dave.txt'
SERVICE ROOT '/odata/dave'
AUTHENTICATION USER dave
USING 'ReadOnly=true;RepeatRequestForDays=10';
```

If you then want to temporarily disable this OData Producer, execute the following statement:

```
ALTER ODATA PRODUCER dave_producer NOT ENABLED;
```

To remove the read-only restriction but retain the repeatable request limit, and re-enable the OData Producer, execute the following statement:

```
ALTER ODATA PRODUCER dave_producer
ENABLED
USING 'ReadOnly=false;RepeatRequestForDays=10';
```

Related Information

OData Support
OData Server Security Considerations
Network Protocol Options
CREATE ODATA PRODUCER Statement [page 908]
DROP ODATA PRODUCER Statement [page 1108]
COMMENT Statement [page 808]

-xs Database Server Option
SYSODATAPRODUCER System View [page 1929]

1.4.4.14 ALTER PROCEDURE Statement

Modifies a procedure.

Syntax

Change the definition of a procedure

```
ALTER PROCEDURE [ owner.]procedure-name procedure-definition
```

```
procedure-definition : see the CREATE PROCEDURE statement
```

Obfuscate a procedure definition

```
ALTER PROCEDURE [ owner.]procedure-name
SET HIDDEN
```

Recompile a procedure

```
ALTER PROCEDURE [ owner.]procedure-name
RECOMPILE
```

Remarks

The ALTER PROCEDURE statement must include the entire new procedure. You can use PROC as a synonym for PROCEDURE.

Change the definition of a procedure

The ALTER PROCEDURE statement is identical in syntax to the CREATE PROCEDURE statement except for the first word. Both Watcom and Transact-SQL dialect procedures can be altered through the use of ALTER PROCEDURE.

With ALTER PROCEDURE, existing privileges on the procedure are not changed. If you execute DROP PROCEDURE followed by CREATE PROCEDURE, execute privileges are reassigned.

Obfuscate a procedure definition

Use SET HIDDEN to obfuscate the definition of the associated procedure and cause it to become unreadable. The procedure can be unloaded and reloaded into other databases.

If SET HIDDEN is used, debugging using the debugger does not show the procedure definition, and the definition is not available through procedure profiling.

> **i Note**
>
> *This change is irreversible.* It is recommended that you retain the original procedure definition outside of the database.

Recompile a procedure

Use the RECOMPILE syntax to recompile a stored procedure. When you recompile a procedure, the definition stored in the catalog is re-parsed and the syntax is verified. For procedures that generate a result set but do not include a RESULT clause, the database server attempts to determine the result set characteristics for the procedure and stores the information in the catalog. This can be useful if a table referenced by the procedure has been altered to add, remove, or rename columns since the procedure was created.

The procedure definition is not changed by recompiling. You can recompile procedures with definitions hidden with the SET HIDDEN clause, but their definitions remain hidden.

> **i** Note
>
> For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must be the owner of the procedure or have one of the following privileges:

- ALTER ANY PROCEDURE system privilege
- ALTER ANY OBJECT system privilege

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

ALTER PROCEDURE is optional ANSI/ISO SQL Language Feature F381. However, in the SQL standard, ALTER PROCEDURE cannot be used to re-define a stored procedure definition, and Transact-SQL dialect procedures are not supported. The ANSI/ISO SQL Standard does not include support for SET HIDDEN or RECOMPILE.

Example

1. The following example creates and then alters a procedure using a variable in the NAMESPACE clause
   1. The following statements create a variable for a NAMESPACE clause:

      ```
      CREATE VARIABLE @ns LONG VARCHAR
      SET @ns = 'http://wsdl.domain.com/';
      ```

   2. The following statement creates a procedure named FtoC that uses a variable in the NAMESPACE clause:

      ```
      CREATE PROCEDURE FtoC ( IN temperature LONG VARCHAR )
      URL 'http://localhost:8082/FtoCService'
      TYPE 'SOAP:DOC'
      NAMESPACE @ns;
      ```

   3. The following statement alters the procedure FtoC so that the temperature parameter accepts a FLOAT data type:

      ```
      ALTER PROCEDURE FtoC ( IN temperature FLOAT )
      URL 'http://localhost:8082/FtoCService'
      NAMESPACE @ns;
      ```

Related Information

Hiding the Contents of a Procedure, Function, Trigger, Event, or View
CREATE PROCEDURE Statement [page 913]
CREATE PROCEDURE Statement [External Call] [page 921]
CREATE PROCEDURE Statement [Web Service] [page 931]
ALTER FUNCTION Statement [page 682]
DROP PROCEDURE Statement [page 1109]

1.4.4.15 ALTER PUBLICATION Statement [MobiLink] [SQL Remote]

Alters a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, a publication identifies replicated data in both consolidated and remote databases.

Syntax

```
ALTER PUBLICATION [ owner.]publication-name alterpub-clause, ...
```

```
alterpub-clause :
ADD article-definition
| ALTER article-definition
| { DELETE | DROP } TABLE [ owner.]table-name
| RENAME publication-name
```

```
article-definition :
TABLE table-name [ ( column-name, ... ) ]
[ WHERE search-condition ]
[ SUBSCRIBE BY expression ]
[ USING ( [PROCEDURE ] [ owner.][procedure-name ]
  FOR UPLOAD { INSERT | DELETE | UPDATE }, ... ) ]
```

Remarks

This statement is applicable only to MobiLink and SQL Remote.

The contribution to a publication from one table is called an article. Changes can be made to a publication by adding, modifying, or deleting articles, or by renaming the publication. If an article is modified, the entire definition of the modified article must be entered.

It is recommended that you perform a successful synchronization of a publication immediately before you alter it.

You cannot use the WHERE clause for publications that are defined as FOR DOWNLOAD ONLY or WITH SCRIPTED UPLOAD.

The SUBSCRIBE BY clause applies to SQL Remote only.

The USING clause is for scripted upload only.

You set options for a MobiLink publication with the ADD OPTION clause in the ALTER SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION SUBSCRIPTION statement.

When altering a MobiLink publication, an article can only be dropped after the execution of a START SYNCHRONIZATION SCHEMA CHANGE statement.

Requires exclusive access to all tables referred to in the statement as well as all tables in publication being modified.

Privileges

You must be the owner of the publication, or have one of the following privileges:

- ALTER privilege on the publication
- SYS_REPLICATION_ADMIN_ROLE system role

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement adds the Customers table to the pub_contact publication.

```
ALTER PUBLICATION pub_contact
   ADD TABLE GROUPO.Customers;
```

Related Information

Publications
Publications and Articles
CREATE PUBLICATION Statement [MobiLink] [SQL Remote] [page 946]
DROP PUBLICATION Statement [MobiLink] [SQL Remote] [page 1111]
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 737]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 997]
SYSSYNC System View [page 1948]
START SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink] [page 1419]
Changes to Avoid on a Running System

1.4.4.16 ALTER REMOTE [MESSAGE] Statement [SQL Remote]

Changes the publisher's message system, or the publisher's address for a given message system, for a message type that has been created.

Syntax

```
ALTER REMOTE [ MESSAGE ]
TYPE message-system
[ ADDRESS address-string ]
```

```
message-system : FILE | FTP | HTTP | SMTP
```

```
address-string : string
```

Parameters

message-system

One of the message systems supported by SQL Remote. It must be one of the following values: FILE, FTP, HTTP, or SMTP.

address

A string containing a valid address for the specified message system.

Remarks

The statement changes the publisher's address for a given message type.

The Message Agent sends outgoing messages from a database by one of the supported message links. The Extraction utility uses this address when it executes the GRANT CONSOLIDATE statement in the remote database.

The address is the publisher's address under the specified message system. If it is an email system, the address string must be a valid email address. If it is a file-sharing system, the address string is a subdirectory of the directory specified by the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement changes the publisher's address for the FILE message link to remote_path.

```
ALTER REMOTE MESSAGE TYPE FILE ADDRESS 'remote_path';
```

The following statement changes the publisher's address for the SMTP message link to user@sample.com.

```
ALTER REMOTE MESSAGE TYPE SMTP ADDRESS 'user@sample.com';
```

Related Information

SQL Remote Message Systems
Altering a Message Type (SQL Central)
CREATE REMOTE [MESSAGE] Statement [SQL Remote] [page 950]
GRANT CONSOLIDATE Statement [SQL Remote] [page 1199]
SQLREMOTE Environment Variable

1.4.4.17 ALTER ROLE Statement

Migrates a compatibility role to a user-defined role, and then drops the compatibility role.

Syntax

```
ALTER ROLE compatibility-role-name
MIGRATE TO new-role-name [, new-sa-role-name, new-sso-role-name ]
```

Parameters

compatibility-role-name

Use this parameter to specify the name of the compatibility role you are migrating.

new-role-name

Use this parameter to specify the name of the new role you are creating.

new-sa-role-name

Use this parameter to specify the name of the new role to migrate the SYS_AUTH_SA_ROLE role to. This parameter is required when migrating SYS_AUTH_DBA_ROLE, which causes SYS_AUTH_SA_ROLE to be migrated automatically.

new-sso-role-name

Use this parameter to specify the name of the new role to migrate the SYS_AUTH_SSO_ROLE role to. This parameter is required when migrating SYS_AUTH_DBA_ROLE, which causes SYS_AUTH_SSO_ROLE to be migrated automatically.

Remarks

The name of the new role must not begin and end with 'SYS_' and '_ROLE', respectively. For example SYS_MyBackup_ROLE is not an acceptable name for a user-defined role, whereas MyBackup_ROLE and SYS_MyBackup are acceptable.

When you execute the ALTER ROLE statement, grantees of the compatibility role are granted the new role.

You can restore migrated compatibility roles that have been migrated and then dropped by executing a CREATE ROLE statement and specifying the compatibility role name. For example, `CREATE ROLE SYS_AUTH_BACKUP_ROLE;` restores the SYS_AUTH_BACKUP_ROLE compatibility role.

Initially, only users with full administration rights (DBAs) can administer the new role, but you can use the CREATE ROLE statement with the OR REPLACE clause to specify additional administrators.

Use the GRANT or REVOKE statements to grant system privileges to the role, or revoke system privileges from the role.

You can migrate the SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE compatibility roles by migrating SYS_AUTH_DBA_ROLE compatibility, which causes SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE to be migrated automatically. When migrating SYS_AUTH_DBA_ROLE, you must include the `new-sa-role-name` and `new-sso-role-name` parameters to give new names to migrated SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE roles.

Privileges

You must have the MANAGE ROLES system privilege and administrative rights on the compatibility role you are migrating.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement migrates all users and underlying system privileges granted to SYS_AUTH_BACKUP_ROLE role to a new role, custom_Backup_ROLE, and then drops SYS_AUTH_BACKUP_ROLE from the database.

```
ALTER ROLE SYS_AUTH_BACKUP_ROLE
MIGRATE TO custom_Backup_ROLE;
```

The following statement migrates all users, underlying system privileges, and roles granted to SYS_AUTH_DBA_ROLE compatibility role to a new role, custom_DBA. It then automatically migrates all users, underlying system privileges, and roles granted to SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE to new roles called custom_SA and custom_SSO, respectively. Finally, it drops SYS_AUTH_DBA_ROLE, SYS_AUTH_SA_ROLE, and SYS_AUTH_SSO_ROLE from the database.

```
ALTER ROLE SYS_AUTH_DBA_ROLE
MIGRATE TO custom_DBA, custom_SA, custom_SSO;
```

Related Information

Roles
CREATE ROLE Statement [page 952]
min_role_admins Option
REVOKE ROLE Statement [page 1352]

1.4.4.18 ALTER SEQUENCE Statement

Alters a sequence.

Syntax

```
ALTER SEQUENCE [owner.]sequence-name
[ RESTART WITH signed-integer ]
[ INCREMENT BY signed-integer ]
[ MINVALUE signed-integer | NO MINVALUE ]
[ MAXVALUE signed-integer | NO MAXVALUE ]
[ CACHE integer | NO CACHE ]
[ CYCLE | NO CYCLE ]
```

Parameters

RESTART WITH clause

Restarts the named sequence with the specified value.

INCREMENT BY clause

Defines the amount the next sequence value is incremented from the last value assigned. The default is 1. Specify a negative value to generate a descending sequence. An error is returned if the INCREMENT BY value is 0.

MINVALUE clause

Defines the smallest value generated by the sequence. The default is 1. An error is returned if MINVALUE is greater than (2^63-1) or less than -(2^63-1). An error is also returned if MINVALUE is greater than MAXVALUE.

MAXVALUE clause

Defines the largest value generated by the sequence. The default is 2^63-1. An error is returned if MAXVALUE is greater than 2^63-1 or less than -(2^63-1).

CACHE clause

Specifies the number of preallocated sequence values that are kept in memory for faster access. When the cache is exhausted, the sequence cache is repopulated and a corresponding entry is written to the transaction log. At checkpoint time, the current value of the cache is forwarded to the ISYSSEQUENCE system table. The default is 100.

CYCLE clause

Specifies whether values should continue to be generated after the maximum or minimum value is reached.

Remarks

If the named sequence cannot be located, an error message is returned.

Privileges

You must be the owner of the sequence, or have one of the following privileges:

- ALTER ANY SEQUENCE system privilege
- ALTER ANY OBJECT system privilege

Side Effects

None

Standards

ANSI/ISO SQL Standard

The ALTER SEQUENCE statement is part of optional ANSI/ISO SQL Language Feature T176. The CACHE clause is not in the standard.

Example

The following example sets a new maximum value for a sequence named Test:

```
ALTER SEQUENCE Test
   MAXVALUE 1500;
```

Related Information

Use of a Sequence to Generate Unique Values
CREATE SEQUENCE Statement [page 959]
DROP SEQUENCE Statement [page 1119]

1.4.4.19 ALTER SERVER Statement

Modifies the attributes of a remote server.

Syntax

Alter a remote server

```
ALTER [ REMOTE ] SERVER server-name
[ CLASS server-class | variable ]
[ USING connection-string-info | variable ]
[ CAPABILITY cap-name { ON | OFF | VALUE variable } ]
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ DEFAULT LOGIN string | variable [ IDENTIFIED BY string | variable ]  | NO
DEFAULT LOGIN ]
```

```
server-class-string :
{ 'ADSODBC' | 'ADS_ODBC' }
| 'ASEODBC' | 'ASE_ODBC' }
| 'DB2ODBC' | 'DB2_ODBC' }
| 'HANAODBC' | 'HANA_ODBC' }
| 'IQODBC' | 'IQ_ODBC' }
| 'MIRROR'
| 'MSACCESSODBC' | 'MSACCESS_ODBC' }
| 'MSSODBC' | 'MSS_ODBC' }
| 'MYSQLODBC' | 'MYSQL_ODBC' }
| 'ODBC'
| 'ORAODBC' | 'ORA_ODBC' }
| 'SAODBC' | 'SA_ODBC' }
| 'ULODBC' | 'UL_ODBC' }
```

```
connection-info-string :
```

```
  { 'data-source-name' | 'connection-string' }
```

Alter a remote server (SAP HANA syntax)

```
ALTER REMOTE SOURCE remote-source-name
    ADAPTER adapter-name | variable
    CONFIGURATION connection-info-string | variable
    [ CAPABILITY cap-name { ON | OFF | VALUE variable } ]
    [ READ ONLY [ ON | OFF | VALUE variable ] ]
    [ WITH CREDENTIAL TYPE { 'PASSWORD' | variable } USING { 'USER=remote-
user;password=remote-password' | variable } | WITH NO CREDENTIAL ]
```

```
connection-info-string :
  { 'data-source-name' | 'connection-string' }
```

Alter a directory access server

```
ALTER SERVER server-name
[ CLASS 'DIRECTORY' ]
[ USING using-string | variable ]
[ CAPABILITY cap-name { ON | OFF | VALUE variable } ]
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ ALLOW { 'ALL' | 'SPECIFIC' | variable } USERS ]
```

```
using-string :
  'ROOT= path [ ;SUBDIRS= n ] [ ;CREATEDIRS= { YES | NO } ] [ ;DELIMITER=
{ / | \ } ]'
```

Parameters

ALTER [REMOTE] SERVER

The REMOTE keyword is optional when altering a remote server and is provided for compatibility with other databases.

CLASS clause Specify this clause to change the server class.

USING clause

Specify the server connection information.

The string in the USING clause can contain local or global variable names enclosed in braces ({`variable-name`}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains `'DSN={@mydsn}'` indicates that `@mydsn` is a SQL variable and that the current value of the `@mydsn` variable is substituted when a connection is made to the remote data access server.

CAPABILITY clause

Specify this clause to turn a server capability ON or OFF. Server capabilities are stored in the ISYSCAPABILITY system table. The names of these capabilities are accessible via the SYSCAPABILITYNAME system view. The ISYSCAPABILITY system table and SYSCAPABILITYNAME system view are not populated with data until the first connection to a remote server is made. For subsequent connections, the database server's capabilities are obtained from the ISYSCAPABILITY system table.

In general, you do not need to alter a server's capabilities. It may be necessary to alter the capabilities of a generic server of class ODBC.

READ ONLY clause (remote server)

Specifies whether the remote server is accessed in read-only mode.

If the READ ONLY clause is not specified, then the read-only setting of the server remains unaffected. If READ ONLY or READ ONLY ON is specified, then the server is changed to be read only. If READ ONLY OFF is specified, then the server is changed to be read-write.

READ ONLY clause (directory access server)

Specifies whether the files accessed by the directory are read-only and cannot be modified. By default, READ ONLY is set to NO.

ALLOW USERS clause

Specifies whether users can use the directory access server without requiring an external login (externlogin). Specifying the ALLOW 'ALL' USERS clause allows you to create or alter directory access servers to no longer require external logins for each user. Specifying the ALLOW 'SPECIFIC' USERS is equivalent to requiring an external login for each user that uses the directory access server. Not specifying this clause is equivalent to specifying ALLOW 'SPECIFIC' USERS.

DEFAULT LOGIN clause

Specify DEFAULT LOGIN to add or change the default login for the remote server. Specifying NO DEFAULT LOGIN removes the default login for the remote server, if one exists.

ALTER REMOTE SOURCE (SAP HANA syntax)

This syntax, including the parameters and their values, is semantically equivalent to the syntax for altering a remote server (CREATE [REMOTE]) syntax, and is provided for compatibility with SAP HANA servers. There is a one-to-one clause match between the two syntaxes as follows:

ADAPTER adapter-name

See the CLASS clause description for the ALTER [REMOTE] SERVER syntax.

CONFIGURATION connection-info-string

See the USING clause description for the ALTER [REMOTE] SERVER syntax.

CAPABILITY cap-name

See the CAPABILITY clause description for the ALTER [REMOTE] SERVER syntax.

READ ONLY [ON | OFF] clause

See the READ ONLY clause description for the ALTER [REMOTE] SERVER syntax (remote server).

WITH CREDENTIAL TYPE clause

See the DEFAULT LOGIN clause description for the ALTER [REMOTE] SERVER syntax. If you are using variables in the WITH CREDENTIAL TYPE clause, then the variable must be a string containing the value 'PASSWORD' and the USING variable must be string using the 'user=...;password=...' format.

Remarks

Changes do not take effect until the next connection to the remote server.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

> **i** Note
>
> For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must have the SERVER OPERATOR system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example changes the DSN of the Adaptive Server Enterprise server named ase_prod to ase_datasource.

```
ALTER SERVER ase_prod
    USING 'ase_datasource';
```

The following example changes the DSN of the Adaptive Server Enterprise server named ase_prod such that its data source name is obtained from the variable ase_source.

```
ALTER SERVER ase_prod
    USING '{ase_source}';
CREATE VARIABLE ase_source VARCHAR(128);
    SET ase_source = 'ase_datasource';
```

The following example changes a capability of a server ase_prod.

```
ALTER SERVER ase_prod
```

```
    CAPABILITY 'insert select' OFF;
```

The following example alters a directory access server so that it retrieves nine levels of subdirectories within the directory `c:\temp`.

```
ALTER SERVER ase_prod
    CLASS 'DIRECTORY'
    USING 'ROOT=c:\\temp;SUBDIRS=9';
```

Related Information

Remote Data Access
Remote Servers and Remote Table Mappings
Directory Access Servers
CREATE SERVER Statement [page 962]
DROP REMOTE CONNECTION Statement [page 1112]
DROP SERVER Statement [page 1120]
SYSCAPABILITY System View [page 1899]
SYSCAPABILITYNAME System View [page 1900]

1.4.4.20 ALTER SERVICE Statement [HTTP Web Service]

Alters an existing HTTP web service.

Syntax

```
ALTER SERVICE service-name
[ TYPE { 'RAW' | 'HTML' | 'JSON' | 'XML' } ]
[ URL [ PATH ] { ON | OFF | ELEMENTS } ]
[ common-attributes ]
[ AS { statement | NULL } ]
```

```
common-attributes :
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

```
method :
DEFAULT
| POST
| GET
| HEAD
| PUT
| DELETE
| NONE
| *
```

Parameters

service-name

Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

You can name your service root, but this name has a special function.

TYPE clause

Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

'RAW'

The result set is sent to the client without any formatting. Utilization of this service requires that all content markup is explicitly provided. Complex dynamic content containing current content with markup, JavaScript and images can be generated on demand. The media type may be specified by setting the Content-Type response header using the sa_set_http_header procedure. Setting the Content-Type header to 'text/html' is good practice when generating HTML markup to ensure that all browsers display the markup as HTML and not text/plain.

'HTML'

The result set is returned as an HTML representation of a table or view.

'JSON'

The result set is returned in JavaScript Object Notation (JSON). A JSON service does not automatically process JSON input. It only presents data (in the response) in JSON format. JSON accepts POST/PUT methods where **`application/x-www-form-urlencoded`** is supported. If for a POST/PUT method, **`Content-Type: application/json`** is specified, then the application may use http_variable('body') to retrieve the JSON (request) content. The database server does not parse the JSON input automatically. It is up to the application to parse it.

'XML'

The result set is returned as XML. If the result set is already XML, no additional formatting is applied. Otherwise, it is automatically formatted as XML. As an alternative approach, a RAW service could return a select using the FOR XML RAW clause having set a valid Content-Type such as text/xml using sa_set_http_header procedure.

URL clause

Determines whether URL paths are accepted and, if so, how they are processed. Specifying URL PATH has the same effect as URL.

OFF

Indicates that the service name in a URL request must not be followed by a path. OFF is the default setting. For example, the following form is disallowed due to the path elements `/aaa/bbb/ccc`.

```
http://host-name/service-name/aaa/bbb/ccc
```

Suppose that `CREATE SERVICE echo URL PATH OFF` was specified when creating the web service. A URL similar to `http://localhost/echo?id=1` produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_HEADER('@HttpQueryString')	id=1

ON

Indicates that the service name in a URL request can be followed by a path. The path value is returned by querying a dedicated HTTP variable called URL. A service can be defined to explicitly provide the URL parameter or it may be retrieved using the HTTP_VARIABLE function. For example, the following form is allowed:

```
http://host-name/service-name/aaa/bbb/ccc
```

Suppose that `CREATE SERVICE echo URL PATH ON` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_VARIABLE('URL')	one/two
HTTP_HEADER('@HttpQueryString')	id=1

ELEMENTS

Indicates that the service name in a URL request may be followed by a path. The path is obtained in segments by specifying a single parameter keyword URL1, URL2, and so on. Each parameter may be retrieved using the HTTP_VARIABLE or NEXT_HTTP_VARIABLE functions. These iterator functions can be used in applications where a variable number of path elements can be provided. For example, the following form is allowed:

```
http://host-name/service-name/aaa/bbb/ccc
```

Suppose that `CREATE SERVICE echo URL PATH ELEMENTS` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_VARIABLE('URL1')	one
HTTP_VARIABLE('URL2')	two
HTTP_VARIABLE('URL3')	NULL
HTTP_HEADER('@HttpQueryString')	id=1

Up to 10 elements can be obtained. A NULL value is returned if the corresponding element is not supplied. In the above example, `HTTP_VARIABLE('URL3')` returns NULL because no corresponding element was supplied.

AUTHORIZATION clause

Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for all services and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) that obfuscates the user and password information using base-64 encoding. Use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses

Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a `404 Not Found` HTTP status.

METHODS clause

Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, PUT, DELETE, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods which are default request types for the RAW, HTML, and XML service types. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	all	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	RAW, HTML, JSON, XML	Enabled by default.
GET	RAW, HTML, JSON, XML	Enabled by default.
HEAD	RAW, HTML, JSON, XML	Enabled by default.
PUT	RAW, HTML, JSON, XML	Not enabled by default.
DELETE	RAW, HTML, JSON, XML	Not enabled by default.
NONE	all	Use NONE to disable access to a service.
*	RAW, HTML, JSON, XML	Same as specifying 'POST,GET,HEAD'.

For example, you can use either of the following clauses to specify that a service supports all HTTP method types:

```
METHODS '*,PUT,DELETE'
METHODS 'POST,GET,HEAD,PUT,DELETE'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause

Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause

Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a `401 Authorization Required` HTTP response status. Based on this response, the web browser prompts for a user ID and password.

> ⚠ Caution
>
> It is strongly recommended that you specify a USER clause when authorization is enabled (default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement

Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

An HTTP request to a non-DISH service with no `statement` specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use `statement` to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following `statement` demonstrates a procedure call that passes two host variables to a procedure called AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

The ALTER SERVICE statement modifies the attributes of a web service.

Privileges

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example demonstrates how to disable an existing web service using the ALTER SERVICE statement:

```
CREATE SERVICE WebServiceTable
   TYPE 'RAW'
   AUTHORIZATION OFF
   USER DBA
   AS SELECT *
      FROM SYS.SYSTAB;
ALTER SERVICE WebServiceTable DISABLE;
```

Related Information

The Database Server as an HTTP Web Server
How to Create and Customize a Root Web Service
How to Develop Web Service Applications in an HTTP Web Server
Tutorial: Using a Database Server to Access a SOAP/DISH Service
CREATE SERVICE Statement [HTTP Web Service] [page 969]
DROP SERVICE Statement [page 1122]
sp_parse_json System Procedure [page 1799]
SYSWEBSERVICE System View [page 1972]
-xs Database Server Option
sa_set_http_header System Procedure [page 1711]
Identifiers [page 6]

ROW Constructor [Composite] [page 534]
ARRAY Constructor [Composite] [page 243]

1.4.4.21 ALTER SERVICE Statement [SOAP Web Service]

Alters an existing SOAP or DISH service over HTTP.

Syntax

SOAP over HTTP services

```
ALTER SERVICE service-name
[ TYPE 'SOAP' ]
[ DATATYPE { ON | OFF | IN | OUT } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
[ common-attributes ]
[ AS statement ]
```

```
common-attributes :
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

```
method :
DEFAULT
| POST
| HEAD
| NONE
```

DISH services

```
ALTER SERVICE service-name
[ TYPE 'DISH' ]
[ GROUP { group-name | NULL } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ]| 'XML' | NULL } ]
[ common-attributes ]
```

```
common-attributes :
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

```
method :
DEFAULT
| POST
| GET
| HEAD
| NONE
| *
```

Parameters

The descriptions of the ALTER SERVICE clauses are identical to those of the CREATE SERVICE statement.

Parameters

service-name

Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

Unlike other services, you cannot use a slash (/) anywhere in a DISH service name.

You can name your service root, but this name has a special function.

TYPE clause

Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

'SOAP'

The result set is returned as an XML payload known as a SOAP envelope. The format of the data may be further refined using by the FORMAT clause. A request to a SOAP service must be a valid SOAP request, not just a general HTTP request. For more information about the SOAP standards, see Simple Object Access Protocol (SOAP) .

'DISH'

A DISH service (Determine SOAP Handler) is a SOAP endpoint that references any SOAP service within its GROUP context. It also exposes the interfaces to its SOAP services by generating a WSDL (Web Services Description Language) for consumption by SOAP client toolkits.

GROUP clause

A DISH service without a GROUP clause exposes all SOAP services defined within the database. By convention, the SOAP service name can be composed of a GROUP and a NAME element. The name is delimited from the group by the last slash character. For example, a SOAP service name defined as 'aaa/bbb/ccc' is 'ccc', and the group is 'aaa/bbb'. Delimiting a DISH service using this convention is invalid. Instead, a GROUP clause is applied to specify the group of SOAP services for which it is to be the SOAP endpoint.

> **i** Note
>
> Slashes are converted to underscores within the WSDL to produce valid XML. Use caution when using a DISH service that does not specify a GROUP clause such that it exposes all SOAP services that may contain slashes. Use caution when using groups with SOAP service names that contain underscores to avoid ambiguity.

DATATYPE clause

Applies to SOAP services only. When DATATYPE OFF is specified, SOAP input parameters and response data are defined as XMLSchema string types. In most cases, true data types are preferred because it

negates the need for the SOAP client to cast the data prior to computation. Parameter data types are exposed in the schema section of the WSDL generated by the DISH service. Output data types are represented as XML schema type attributes for each column of data.

The following values are permitted for the DATATYPE clause:

ON

Generates data typing of input parameters and result set responses.

OFF

All input parameters and response data are typed as XMLSchema string (default).

IN

Generates true data types for input parameters only. Response data types are XMLSchema string.

OUT

Generates true data types for responses only. Input parameters are typed as XMLSchema string.

FORMAT clause

This clause specifies the output format when sending responses to SOAP client applications.

The SOAP service format is dictated by the associated DISH service format specification when it is not specified by the SOAP service. The default format is DNET.

SOAP requests should be directed to the DISH service (the SQL Anywhere SOAP endpoint) to leverage common formatting rules for a group of SOAP services (SOAP operations). A SOAP service FORMAT specification overrides that of a DISH service. The format specification of the DISH service is used when a SOAP service does not define a FORMAT clause. If no FORMAT is provided by either service then the default is 'DNET'.

The following formats are supported:

'DNET'

The output is in a System.Data.DataSet compatible format for consumption by .NET client applications. (default)

'CONCRETE'

This output format is used to support client SOAP toolkits that are capable of generating interfaces representing arrays of row and column objects but are not able to consume the DNET format. Java and .NET clients can easily consume this output format.

The specific format is exposed within the WSDL as an explicit dataset object or a SimpleDataset. Both dataset representations describe a data structure representing an array of rows with each row containing an array of columns. An explicit dataset object has the advantage of representing the actual shape of the result set by providing parameter names and data types for each column in the row. In contrast, the SimpleDataset exposes rows containing an unbounded number of columns of any type.

FORMAT 'CONCRETE' EXPLICIT ON requires that the Service statement calls a stored procedure which defines a RESULT clause. Having met this condition, the SOAP service will expose an explicit dataset whose name begins with the service name appended with Dataset.

If the condition is not met, a SimpleDataset is used.

'XML'

The output is generated in an XMLSchema string format. The response is an XML document that requires further processing by the SOAP client to extract column data. This format is suitable for SOAP clients that cannot generate intermediate interface objects that represent arrays of rows and columns.

NULL

A NULL type causes the SOAP or DISH service to use the default behavior. The format type of an existing service is overwritten when using the NULL type in an ALTER SERVICE statement.

AUTHORIZATION clause

Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for SOAP services, and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) which obfuscates the user and password information using base-64 encoding. It is recommended that you use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses

Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a `404 Not Found` HTTP status.

METHODS clause

Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods. The default method types for SOAP services are POST and HEAD. The default method types for DISH services are GET, POST, and HEAD. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	both	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	both	Enabled by default for SOAP.
GET	DISH only	Enabled by default for DISH.
HEAD	both	Enabled by default for SOAP and DISH.

Method value	Applies to service	Description
NONE	both	Use NONE to disable access to a service. When applied to a SOAP service, the service cannot be directly accessed by a SOAP request. This enforce exclusive access to a SOAP operation through a DISH service SOAP endpoint. It is recommended that you specify `METHODS 'NONE'` for each SOAP service.
*	DISH only	Same as specifying `'POST,GET,HEAD'`.

For example, you can use the following clause to specify that a service supports all SOAP over HTTP method types:

```
METHODS 'POST,HEAD'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause

Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause

Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a `401 Authorization Required` HTTP response status. Based on this response, the web browser prompts for a user ID and password.

> ⚠ Caution
>
> It is strongly recommended that you specify a USER clause when authorization is enabled (the default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement

Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

A DISH service is the only service that must either define a null statement, or not define a statement. A SOAP service must define a statement. Any other SERVICE can have a NULL statement, but only if it is configured with AUTHORIZATION ON.

An HTTP request to a non-DISH service with no `statement` specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use `statement` to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following `statement` demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

The ALTER SERVICE statement modifies the attributes of a web service.

Privileges

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example demonstrates how to disable an existing web service using the ALTER SERVICE statement:

```
CREATE SERVICE WebServiceTable
   TYPE 'SOAP'
   AUTHORIZATION OFF
```

```
   USER DBA
   AS SELECT *
      FROM SYS.SYSTAB;
ALTER SERVICE WebServiceTable DISABLE;
```

Related Information

The Database Server as an HTTP Web Server
How to Create and Customize a Root Web Service
Tutorial: Using a Database Server to Access a SOAP/DISH Service
CREATE SERVICE Statement [SOAP Web Service] [page 975]
DROP SERVICE Statement [page 1122]
SYSWEBSERVICE System View [page 1972]
-xs Database Server Option

1.4.4.22 ALTER SPATIAL REFERENCE SYSTEM Statement

Changes the settings of an existing spatial reference system. See the Remarks section for considerations before altering a spatial reference system.

Syntax

```
ALTER SPATIAL REFERENCE SYSTEM
srs-name
[ srs-attribute [ srs-attribute ... ] ]
```

```
srs-name : string
```

```
srs-attribute :
SRID srs-id
| DEFINITION { definition-string | NULL }
| ORGANIZATION { organization-name IDENTIFIED BY organization-srs-id | NULL }
| TRANSFORM DEFINITION { transform-definition-string | NULL }
| LINEAR UNIT OF MEASURE linear-unit-name
| ANGULAR UNIT OF MEASURE { angular-unit-name | NULL }
| TYPE { ROUND EARTH | PLANAR }
| COORDINATE coordinate-name { UNBOUNDED | BETWEEN low-number AND high-
number }
| ELLIPSOID SEMI MAJOR AXIS semi-major-axis-length { SEMI MINOR AXIS semi-minor-
axis-length | INVERSE FLATTENING inverse-flattening-ratio }
| SNAP TO GRID { grid-size | DEFAULT }
| TOLERANCE { tolerance-distance | DEFAULT }
| AXIS ORDER axis-order
| POLYGON FORMAT polygon-format
| STORAGE FORMAT storage-format
```

```
srs-id : integer
```

```
semi-major-axis-length : number
```

```
semi-minor-axis-length : number

inverse-flattening-ratio : number

grid-size : DOUBLE : usually between 0 and 1

tolerance-distance : number

axis-order :  { 'x/y/z/m' | 'long/lat/z/m' | 'lat/long/z/m' }

polygon-format : { 'CounterClockWise' | 'Clockwise' | 'EvenOdd' }

storage-format : { 'Internal' | 'Original' | 'Mixed' }
```

Parameters

Complete definitions for each of the clauses is provided in the CREATE SPATIAL REFERENCE SYSTEM statement.

IDENTIFIED BY clause

Use this clause to change the SRID number for the spatial reference system.

DEFINITION clause

Use this clause to set, or override, default coordinate system settings.

ORGANIZATION clause

Use this clause to specify information about the organization that created the spatial reference system that the spatial reference system is based on.

TRANSFORM DEFINITION clause

Use this clause to specify a description of the transform to use for the spatial reference system. Currently, only the PROJ.4 transform is supported.

The transform definition is used by the ST_Transform method when transforming data between spatial reference systems. Some transforms may still be possible even if there is no `transform-definition-string` defined.

COORDINATE clause

Use this clause to specify the bounds on the spatial reference system's dimensions. `coordinate-name` is the name of the coordinate system used by the spatial reference system. For non-geographic types `coordinate-name` can be x, y, or m. For geographic types, `coordinate-name` can be LATITUDE, LONGITUDE, z, or m.

LINEAR UNIT OF MEASURE clause

Use this clause to specify the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure defined in the ST_UNITS_OF_MEASURE consolidated view.

ANGULAR UNIT OF MEASURE clause

Use this clause to specify the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the ST_UNITS_OF_MEASURE consolidated view.

TYPE clause

Use the TYPE clause to control how the spatial reference system interprets lines between points. For geographic spatial reference systems, the TYPE clause can specify either ROUND EARTH (the default) or PLANAR. For non-geographic spatial reference systems, the type must be PLANAR.

ELLIPSOID clause

Use the ellipsoid clause to specify the values to use for representing the Earth as an ellipsoid for spatial reference systems of type ROUND EARTH. If the DEFINITION clause is present, it can specify ellipsoid definition. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

SNAP TO GRID clause

For flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid the database server uses when performing calculations. Specify SNAP TO GRID DEFAULT to set the grid size to the default that the database server would use.

For round-Earth spatial reference systems, SNAP TO GRID must be set to 0.

TOLERANCE clause

For flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points.

For round-Earth spatial reference systems, TOLERANCE must be set to 0.

POLYGON FORMAT clause

Use the POLYGON FORMAT clause to change the polygon interpretation. The following values are supported:

- 'CounterClockwise'
- 'Clockwise'
- 'EvenOdd'

The default polygon format is 'EvenOdd'.

STORAGE FORMAT clause

Use the STORAGE FORMAT clause to control what is stored when spatial data is loaded into the database. Possible values are:

'Internal'

The database server stores only the normalized representation. Specify this when the original input characteristics do not need to be reproduced. This is the default for planar spatial reference systems (TYPE PLANAR).

> **i** Note
>
> If you are using MobiLink to synchronize your spatial data, you should specify **Mixed** instead. MobiLink tests for equality during synchronization, which requires the data in its original format.

'Original'

The database server stores only the original representation. The original input characteristics can be reproduced, but all operations on the stored values must repeat normalization steps, possibly slowing down operations on the data.

'Mixed'

The database server stores the internal version and, if if the internal version is different from the original version, the original version as well. By storing both versions, the original representation characteristics can be reproduced and operations on stored values do not need to repeat normalization steps. However, storage requirements may increase significantly because potentially two representations are being stored for each geometry.

Mixed is the default format for round-Earth spatial reference systems (TYPE ROUND EARTH).

Remarks

You cannot alter a spatial reference system if there is existing data that references it. For example, if you have a column declared as ST_Point(SRID=8743), you cannot alter the spatial reference system with SRID 8743. This is because many spatial reference system attributes, such as storage format, impact the storage format of the data. If you have data that references the SRID, create a new spatial reference system and transform the data to the new SRID.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must be the owner of the spatial reference system, or have one of the following privileges:

- ALTER privilege on the spatial reference system
- MANAGE ANY SPATIAL OBJECT system privilege
- ALTER ANY OBJECT system privilege

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example changes the polygon format of a fictitious spatial reference system named mySpatialRef to EvenOdd.

```
ALTER SPATIAL REFERENCE SYSTEM mySpatialRef
POLYGON FORMAT 'EvenOdd';
```

Related Information

Spatial Data
CREATE SPATIAL REFERENCE SYSTEM Statement [page 981]
DROP SPATIAL REFERENCE SYSTEM Statement [page 1123]
ST_UNITS_OF_MEASURE Consolidated View [page 1979]
ST_Transform Method

1.4.4.23 ALTER STATISTICS Statement

Controls whether statistics are automatically updated on a column, or columns, in a table.

Syntax

```
ALTER STATISTICS
[ ON ] table [ ( column1 [ , column2 ... ] ) ]
AUTO UPDATE { ENABLE | DISABLE }
```

Parameters

ON

The word ON is optional. Including it has no impact on the execution of the statement.

AUTO UPDATE clause

Specify whether to enable or disable automatic updating of statistics for the column(s).

Remarks

During normal execution of queries, DML statements, and LOAD TABLE statements, the database server automatically maintains column statistics for use by the optimizer. The benefit of maintaining statistics for some columns may not outweigh the overhead necessary to generate them. For example, if a column is not

queried often, or if it is subject to periodic mass changes that are eventually rolled back, there is little value in continually updating its statistics. Use the ALTER STATISTICS statement to suppress the automatic updating of statistics for these types of columns.

When automatic updating is disabled, you can still update the statistics for the column using the CREATE STATISTICS and DROP STATISTICS statements. However, you should only update them if it has been determined that it would have a positive impact on performance. Normally, column statistics should not be disabled.

Privileges

You must be the table owner, or have one of the following privileges:

- MANAGE ANY STATISTICS system privilege
- ALTER ANY OBJECT system privilege

Side Effects

If automatic updating has been disabled, the statistics may become out of date. Re-enabling does not immediately bring them up to date. Execute the CREATE STATISTICS statement to recreate them, if necessary.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example disables the automatic updating of statistics on the Street column in the Customers table:

```
ALTER STATISTICS GROUPO.Customers ( Street ) AUTO UPDATE DISABLE;
```

Related Information

CREATE STATISTICS Statement [page 991]
DROP STATISTICS Statement [page 1126]

1.4.4.24 ALTER SYNCHRONIZATION PROFILE Statement [MobiLink]

Changes a SQL Anywhere synchronization profile. Synchronization profiles are named collections of synchronization options that can be used to control synchronization.

Syntax

```
ALTER SYNCHRONIZATION PROFILE name
MERGE string
```

Parameters

name

The name of the synchronization profile to alter.

MERGE clause

Use this clause to change existing, or add new, options to a synchronization profile.

string

A string of one or more synchronization option value pairs, separated by semicolons. For example, `'option1=value1;option2=value2'`.

Remarks

Synchronization profiles define how a SQL Anywhere database synchronizes with the MobiLink server.

When MERGE is used in the ALTER SYNCHRONIZATION PROFILE statement, options specified in the string are added to those already in the synchronization profile. If an option in the string already exists in profile, then the value from the string replaces the value already stored in the profile.

For example, executing the following statements leaves the profile `myProfile` with the value `subscription=s2;verbosity=high;uploadonly=on`.

```
CREATE SYNCHRONIZATION PROFILE myProfile 'subscription=p1;verbosity=high';
ALTER SYNCHRONIZATION PROFILE myProfile MERGE 'subscription=p2;uploadonly=on';
```

When setting extended options, use the following syntax:

```
ALTER SYNCHRONIZATION PROFILE myprofile MERGE
's=mysub;e={ctp=tcpip;adr=''host=localhost;port=2439''}';
```

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

CREATE SYNCHRONIZATION PROFILE Statement [MobiLink] [page 996]
DROP SYNCHRONIZATION PROFILE Statement [MobiLink] [page 1129]

1.4.4.25 ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

Alters the properties of a synchronization subscription in a SQL Anywhere remote database.

Syntax

```
ALTER SYNCHRONIZATION SUBSCRIPTION
{ subscription-name | TO publication-name [ FOR ml-username, ...  ] } { alter-clause ... }
```

```
alter-clause :
RENAME new-subscription-name
| TYPE network-protocol
| ADDRESS protocol-options
| ADD OPTION option=value, ...
| ALTER OPTION option=value, ...
| DELETE { ALL OPTION | OPTION option, ... }
| SET SCRIPT VERSION=script-version
```

```
subscription-name : identifier
```

```
publication-name : identifier
```

```
ml-username : identifier

new-subscription-name : identifier

network-protocol : http | https | tls | tcpip | NULL

protocol-options : string | NULL

value : string | integer

option : identifier

script-version : string | NULL
```

Parameters

TO clause

This clause specifies the name of a publication.

If the TO clause is used without a FOR clause, you cannot use the RENAME or SET SCRIPT VERSION clauses.

FOR clause

This clause specifies one or more MobiLink user names.

Omit the FOR clause to set the protocol type, protocol options, and extended options for a publication.

If the TO clause is used without a FOR clause, you cannot use the RENAME or SET SCRIPT VERSION clauses.

RENAME clause

This clause specifies a new name for the subscription.

If the TO clause is used without a FOR clause, you cannot use the RENAME clause.

TYPE clause

This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause

This clause specifies network protocol options, including the location of the MobiLink server.

ADD OPTION, ALTER OPTION, DELETE OPTION, and DELETE ALL OPTION clauses

These clauses allow you to add, alter, delete, or delete all extended options. You can specify only one option in each clause. No option is specified for Delete All.

The values for each option cannot contain the characters " = " or " , " or " ; ".

SET SCRIPT VERSION clause

This clause specifies the script version to use during synchronization. You can alter the script version without making a schema change.

If the TO clause is used without a FOR clause, you cannot use the SET SCRIPT VERSION clause.

Remarks

The `network-protocol`, `protocol-options`, and `options` can be set in several places.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Depending on the privileges a user has, they may be able to view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example changes the address of the MobiLink server for the sales subscription:

```
ALTER SYNCHRONIZATION SUBSCRIPTION sales
TYPE TCPIP
ADDRESS 'host=10.11.12.132;port=2439';
```

Related Information

Synchronization Subscription Creation
MobiLink SQL Anywhere Client Extended Options

How dbmlsync Resolves Conflicting Options
Script Versions
CREATE PUBLICATION Statement [MobiLink] [SQL Remote] [page 946]
DROP PUBLICATION Statement [MobiLink] [SQL Remote] [page 1111]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 997]
SYSSYNC System View [page 1948]
MobiLink Client Network Protocol Options
CommunicationType (ctp) Extended Option
MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax

1.4.4.26 ALTER SYNCHRONIZATION USER Statement [MobiLink]

Alters the properties of a MobiLink user in a SQL Anywhere remote database.

Syntax

```
ALTER SYNCHRONIZATION USER ml-username
[ TYPE network-protocol ]
[ ADDRESS protocol-options ]
[ ADD OPTION option=value, ... ]
[ ALTER OPTION option=value, ... ]
[ DELETE { ALL OPTION | OPTION option } ]
```

```
ml-username : identifier
```

```
network-protocol : http | https | tls | tcpip | NULL
```

```
protocol-options : string | NULL
```

```
value : string | integer
```

Parameters

TYPE clause

This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause

This clause specifies `protocol-options` in the form `keyword=value`, separated by semicolons. Which settings you supply depends on the communication protocol you are using (TCP/IP, TLS, HTTP, or HTTPS).

ADD OPTION, ALTER OPTION, DELETE OPTION, and DELETE ALL OPTION clauses

These clauses allow you to add, modify, delete, or delete all extended options. You may specify only one option in each clause. No option is specified for Delete All.

Remarks

The `network-protocol`, `protocol-options`, and `options` can be set in several places.

This statement causes options and other information to be stored in the ISYSSYNC system table. Depending on the privileges a user has, passwords and encryption certificates can be visible. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Requires exclusive access to all tables referred to in the publication.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

MobiLink Users in a Synchronization System
MobiLink SQL Anywhere Client Extended Options
How dbmlsync Resolves Conflicting Options
MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax
CREATE SYNCHRONIZATION USER Statement [MobiLink] [page 1000]
CommunicationType (ctp) Extended Option
DROP SYNCHRONIZATION USER Statement [MobiLink] [page 1132]
SYSSYNC System View [page 1948]
MobiLink Client Network Protocol Options

1.4.4.27 ALTER TABLE Statement

Modifies a table definition or disables dependent views.

Syntax

Altering an existing table

```
ALTER TABLE [owner.]table-name { alter-clause, ... }
```

```
alter-clause :
ADD create-clause
| ALTER column-name column-alteration
| ALTER [ CONSTRAINT constraint-name ] CHECK ( condition )
| DROP drop-object
| RENAME rename-object
| table-alteration
```

```
create-clause :
column-name [ AS ] column-data-type [ new-column-attribute ... ]
| table-constraint
| PCTFREE integer
```

```
column-alteration :
 { column-data-type | alterable-column-attribute } [ alterable-column-
attribute ... ]
| SET COMPUTE ( compute-expression )
| [ SET ] DEFAULT default-value
| ADD [ CONSTRAINT constraint-name ] CHECK ( condition )
| DROP { DEFAULT | COMPUTE | CHECK | CONSTRAINT constraint-name }
```

```
drop-object :
column-name
| CHECK
| CONSTRAINT constraint-name
| UNIQUE ( index-columns-list )
| FOREIGN KEY fkey-name
| PRIMARY KEY
```

```
rename-object :
new-table-name
| column-name TO new-column-name
| CONSTRAINT constraint-name TO new-constraint-name
```

```
table-alteration :
PCTFREE DEFAULT
| [ NOT ] ENCRYPTED
```

```
new-column-attribute :
[ NOT ] NULL
| [ SET ] DEFAULT default-value
| COMPRESSED
| INLINE { inline-length | USE DEFAULT }
| PREFIX { prefix-length | USE DEFAULT }
| [ NO ] INDEX
| IDENTITY
| COMPUTE ( expression )
| column-constraint
```

```
table-constraint :
[ CONSTRAINT constraint-name ] {
   CHECK ( condition )
   | UNIQUE [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC |
DESC ], ... )
   | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC |
DESC ], ... )
   | foreign-key
  }
```

```
column-constraint :
[ CONSTRAINT constraint-name ] {
   CHECK ( condition )
   | UNIQUE [ CLUSTERED | NONCLUSTERED ]
   | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ]
   | REFERENCES table-name [ ( column-name ) ]
      [ MATCH [ UNIQUE ] { SIMPLE | FULL } ]
      [ actions ]
      [ CLUSTERED | NONCLUSTERED ]
   | NOT NULL
  }
```

```
alterable-column-attribute :
[ NOT ] NULL
| DEFAULT default-value
| [ CONSTRAINT constraint-name ] CHECK { NULL | ( condition ) }
| [ NOT ] COMPRESSED
| INLINE { inline-length | USE DEFAULT }
| PREFIX { prefix-length | USE DEFAULT }
| [ NO ] INDEX
```

```
default-value :
 special-value
| string
| global variable
| [ - ] number
| ( constant-expression )
| ( sequence-expression )
| built-in-function( constant-expression )
| AUTOINCREMENT
| GLOBAL AUTOINCREMENT [ ( partition-size ) ]
```

```
special-value :
CURRENT DATABASE
| CURRENT DATE
| CURRENT TIME
| [ CURRENT ] TIMESTAMP
| CURRENT PUBLISHER
| CURRENT REMOTE USER
| [ CURRENT ] USER
| [ CURRENT ] UTC TIMESTAMP
| EXECUTING USER
| INVOKING USER
| LAST USER
| NULL
| PROCEDURE OWNER
| SESSION USER
```

```
foreign-key :
[ NOT NULL ] FOREIGN KEY [ role-name ]
```

```
    [ ( column-name [ ASC | DESC ], ... )
    REFERENCES table-name
    [ ( pkey-column-list ) ]
    [ MATCH [ UNIQUE] { SIMPLE | FULL } ]
    [ actions ]
    [ CHECK ON COMMIT ]
    [ CLUSTERED | NONCLUSTERED ]
    [ FOR OLAP WORKLOAD ]
```

```
actions :
[ ON UPDATE action ] [ ON DELETE action ]
```

```
action :
CASCADE | SET NULL | SET DEFAULT | RESTRICT
```

Disabling view dependencies

```
ALTER TABLE [owner.]table-name {
 DISABLE VIEW DEPENDENCIES
}
```

Altering a table owner

```
ALTER TABLE [owner.]table-name ALTER OWNER TO owner
[ { PRESERVE | DROP } PRIVILEGES ]
[ { PRESERVE | DROP } FOREIGN KEYS ]
```

Parameters

Adding clauses

There are several clauses you can use for adding columns or table constraints to a table:

ADD column-name [AS] column-data-type [new-column-attribute ...] clause

Use this clause to add a new column to the table, specifying the data type and attributes for the column. Define the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in another table or view. When you specify the %TYPE attribute, other attributes on the object referenced in the %TYPE specification, such as default values, constraints, and whether NULLs are allowed, are not part of the definition that is inherited and must be specified separately.

NULL and NOT NULL clauses

Use this clause to specify whether to allow NULLs in the column. By default, new columns allow NULL values. BIT type columns automatically have the NOT NULL constraint applied when they are created, but you can declare a BIT type column to be nullable.

DEFAULT clause

If a DEFAULT value is specified, it is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT value is specified, it is equivalent to DEFAULT NULL.

Following is a list of possible values for DEFAULT:

constant-expression

Constant expressions that do not reference database objects are allowed in a DEFAULT clause, so functions such as GETDATE or DATEADD can be used. If the expression is not a function or simple value, it must be enclosed in parentheses.

global-variable

A global variable.

sequence-expression

Set DEFAULT to the current value or next value from a sequence in the database.

string

A string value.

AUTOINCREMENT

When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type.

On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, that value is inserted and then used as a starting point for subsequent inserts.

Deleting rows does not decrement the AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. After an explicit insert of a column value less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the next value to be generated exceeds the maximum value that can be stored in the column to which the AUTOINCREMENT is assigned, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated.

The next value to use for a column can be reset using the sa_reset_identity procedure.

GLOBAL AUTOINCREMENT

This default is intended for use when multiple databases are used in a MobiLink synchronization environment or SQL Remote replication.

This option is similar to AUTOINCREMENT, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. The database server supplies default values in a database only from the partition uniquely identified by that database's number.

The partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size can be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

If the column is of type BIGINT or UNSIGNED BIGINT, the default partition size is 2^{32} = 4294967296; for columns of all other types, the default partition size is 2^{16} = 65536. Since these defaults may be inappropriate, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

When using this default, the value of the public option global_database_id in each database must be set to a unique, non-negative integer. This value uniquely identifies the database and indicates from which partition default values are to be assigned. The range of allowed values is $n\,p + 1$ to $p(n + 1)$, where n is the value of the public option global_database_id and p is the partition size. For example, if you define the partition size to be 1000 and set global_database_id to 3, then the range is from 3001 to 4000.

If the previous value is less than $p(n + 1)$, the next default value is one greater than the previous largest value in the column. If the column contains no values, the first default value is $n\,p + 1$. Default column values are not affected by values in the column outside the current partition; that is, by numbers less than $np + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink or SQL Remote.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

GLOBAL AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the supply of values within the partition has been exhausted, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated. In this case, a new value of global_database_id should be assigned to the database to allow default values to be chosen from another partition. To detect that the supply of unused values is low and handle this condition, create an event of type GlobalAutoincrement.

Because the public option global_database_id cannot be set to a negative value, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

If the public option global_database_id is set to the default value of 2147483647, a NULL value is inserted into the column. If NULL values are not permitted, attempting to insert the row causes an error.

The next value to use for a column can be reset using the sa_reset_identity procedure.

special-value

You use one of several special values in the DEFAULT clause (for example, CURRENT DATE) including, but not limited to, the following special values.

[CURRENT] TIMESTAMP

Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the current date and time of day whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT TIMESTAMP instead of DEFAULT TIMESTAMP.

Columns declared with DEFAULT TIMESTAMP contain unique values, so that applications can detect near-simultaneous updates to the same row. If the current TIMESTAMP value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate TIMESTAMP values based on the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable @@dbts returns a TIMESTAMP value representing the last value generated for a column using DEFAULT TIMESTAMP.

When the database is using a simulated time zone, the simulated time zone is used to calculate these values.

[CURRENT] UTC TIMESTAMP

Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT UTC TIMESTAMP, a default value is provided for inserts, and the value is updated with the current Coordinated Universal Time (UTC) whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT UTC TIMESTAMP instead of DEFAULT UTC TIMESTAMP.

The behavior of this default is the same as TIMESTAMP and CURRENT TIMESTAMP except that the date and time of day is in Coordinated Universal Time (UTC).

LAST USER

LAST USER is the user ID of the user who last modified the row.

LAST USER can be used as a default value in columns with character data types.

On INSERT, this default has the same effect as CURRENT USER.

On UPDATE, if a column with a default of LAST USER is not explicitly modified, it is changed to the name of the current user.

When used with DEFAULT TIMESTAMP or DEFAULT UTC TIMESTAMP, a default of LAST USER can be used to record (in separate columns) both the user and the date and time a row was last changed.

column-constraint clause

Use this clause to add a constraint to the column.

> **i** Note
>
> With the exception of CHECK constraints, when a new constraint is added, the database server validates existing values to confirm that they satisfy the constraint. CHECK constraints are enforced only for operations that occur after the table alteration is complete.

Possible column constraints include:

CHECK clause

This constraint allows arbitrary conditions to be verified. For example, a CHECK constraint could be used to ensure that a column called Sex only contains the values M or F.

If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

UNIQUE clause

Use this subclause to specify that values in the column must be unique, and whether to create a clustered or nonclustered index.

PRIMARY KEY clause

Use this subclause to make the column a primary key, and specify whether to use a clustered or nonclustered index.

REFERENCES clause

Use this subclause to add or alter a reference to another table, to specify how matches are handled, and to specify whether to use a clustered or nonclustered index.

MATCH clause

Use this subclause to control what is considered a match when using a multi-column foreign key. It also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

NULL and NOT NULL clauses

Use this clause to specify whether to allow NULL values in the column. By default, NULLs are allowed.

COMPRESSED clause

Use this clause to compress the column.

INLINE and PREFIX clauses

The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. In this context, the term BLOB refers to both character and binary column values of any length.

BLOBs that are shorter than or equal to the INLINE value are stored inline, that is, within the row.

BLOBs that are longer than the value specified by the INLINE clause are stored outside the row in table extension pages. In this case, a prefix of the BLOB may be stored inline. The prefix is composed from the leading bytes of the BLOB. Use the PREFIX clause to specify the length of the prefix. A PREFIX value of 0 means that no prefix will be stored. The PREFIX value cannot exceed the INLINE value. The PREFIX clause can affect the performance of requests that use the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The data that is stored inline is not compressed. If a prefix exists and it indicates that the BLOB value will not satisfy the request, then no decompression or the data stored outside the row in table extension pages is necessary.

If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.
- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

> **i Note**
>
> It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set INLINE to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the INLINE nor PREFIX value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an INLINE value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

INDEX and NO INDEX clauses

When storing BLOBs (character or binary types only), specify INDEX to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify NO INDEX.

> **i** Note
>
> A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

IDENTITY clause

The IDENTITY clause is a Transact-SQL-compatible alternative to using DEFAULT AUTOINCREMENT. A column defined with IDENTITY is implemented as DEFAULT AUTOINCREMENT.

COMPUTE clause

When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The COMPUTE clause is ignored for remote tables.

Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.

ADD table-constraint clause

Use this clause to add a table constraint. Table constraints place limits on what data columns in the table can hold. When adding or altering table constraints, the optional constraint name allows you to modify or drop individual constraints. Following is a list of the table constraints you can add.

UNIQUE

Use this subclause to specify that values in the columns specified in `column-list` must be unique, and, optionally, whether to use a clustered or nonclustered index.

PRIMARY KEY

Use this subclause to add or alter the primary key for the table, and specify whether to use a clustered or nonclustered index. The table must not already have a primary key that was created by the CREATE TABLE statement or another ALTER TABLE statement.

foreign-key

Use this subclause to add a foreign key as a constraint. If you use a subclause other than ADD FOREIGN KEY with the ALTER TABLE statement on a table with dependent materialized views, the ALTER TABLE statement fails. For all other clauses, you must disable the dependent materialized views and then re-enable them when your changes are complete.

You can specify a MATCH subclause to control what is considered a match when using a multi-column foreign key. It also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

ADD PCTFREE clause

Specify the percentage of free space you want to reserve in each table page. The free space is used if rows increase in size when the data is updated. If there is no free space in a table page, every increase in the size of a row on that page requires the row to be split across multiple table pages, causing row fragmentation and possible performance degradation. A free space percentage of 0 specifies that no free space is to be left on each page. Each page is to be fully packed. A high free space percentage causes each row to be inserted into a page by itself. If PCTFREE is not set, or is dropped, the default PCTFREE value is applied according to the database page size (200 bytes for a 4 KB or larger page size). The value for PCTFREE is stored in the ISYSTAB system table. When PCTFREE is set, all subsequent inserts into table pages use the new value, but rows that were already inserted are not affected. The value persists until it is changed. The PCTFREE specification can be used for base, global temporary, or local temporary tables.

Altering clauses

There are several clauses you can use to alter the definition for a column or table:

ALTER column-name column-alteration clause

Use this clause to change attributes for the specified column. If a column is contained in a unique constraint, a foreign key, or a primary key, you can change only the default for the column. However, for any other change, you must delete the key or constraint before the column can be modified.

column-data-type clause

Use this clause to alter the length or data type of the column. Specify the new data type, or use the %TYPE attribute to set the data type to the data type of a column in another table or view. If necessary, then the data in the modified column is converted to the new data type. If a conversion error occurs, then the operation fails and the table is left unchanged. You cannot reduce the size of a column. For example, you cannot change a column from a VARCHAR(100) to a VARCHAR(50).

[NOT] NULL clause

Use this clause to change whether NULLs are allowed in the column. If NOT NULL is specified, and the column value is NULL in any of the existing rows, then the operation fails and the table is left unchanged.

CHECK NULL

Use this clause to delete all check constraints for the column.

DEFAULT clause

Use this clause to change the default value for the column.

DEFAULT NULL clause

Use this clause to remove the default value for the column.

[CONSTRAINT constraint-name] CHECK { NULL | (condition) } clause

Use this clause to add a CHECK constraint on the column.

If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

[NOT] COMPRESSED clause

Use this clause to change whether the column is compressed.

INLINE and PREFIX clauses

The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. BLOBs smaller than or equal to the value specified by the INLINE clause are stored within the row. BLOBs that exceed the value specified by the INLINE clause are stored outside the row in table extension pages. Also, a copy of some bytes from the beginning of the BLOB may be kept in the row when a BLOB is larger than the INLINE value. Use the PREFIX clause to specify how many bytes are kept in the row. The PREFIX clause can improve the performance of requests that need the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.

If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.
- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

> **i** Note
>
> It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set INLINE to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the INLINE nor PREFIX value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an INLINE value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

INDEX and NO INDEX clauses

When storing BLOBs (character or binary types only), specify INDEX to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify NO INDEX.

> **i Note**
>
> A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

SET COMPUTE clause

When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The COMPUTE clause is ignored for remote tables.

Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.

ALTER CONSTRAINT constraint-name CHECK clause

Use this clause to alter a named check constraint for the table.

To alter the constraint to specify a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

Dropping clauses

DROP DEFAULT

Drops the default value set for the table or specified column. Existing values do not change.

DROP COMPUTE

Removes the COMPUTE attribute for the specified column. This statement does not change any existing values in the table.

DROP CHECK

Drops all CHECK constraints for the table or specified column. DELETE CHECK is also accepted.

DROP CONSTRAINT constraint-name

Drops the named constraint for the table or specified column. DELETE CONSTRAINT is also accepted.

DROP column-name

Drops the specified column from the table. DELETE `column-name` is also accepted. If the column is contained in any index, unique constraint, foreign key, or primary key, then the index, constraint, or key must be deleted before the column can be deleted. This does not delete CHECK constraints that refer to the column.

DROP UNIQUE (column-name ...)

Drop the unique constraints on the specified column(s). Any foreign keys referencing this unique constraint are also deleted. DELETE UNIQUE (`column-name` ...) is also accepted.

DROP FOREIGN KEY fkey-name

Drop the specified foreign key. DELETE FOREIGN KEY `fkey-name` is also accepted.

DROP PRIMARY KEY

Drop the primary key. All foreign keys referencing the primary key for this table are also deleted. DELETE PRIMARY KEY is also accepted.

Renaming clauses

RENAME new-table-name

Change the name of the table to `new-table-name`. Any applications using the old table name must be modified, as necessary.

RENAME column-name TO new-column-name

Change the name of the column to the `new-column-name`. Any applications using the old column name must be modified, as necessary.

RENAME CONSTRAINT constraint-name TO new-constraint-name

Change the name of the constraint to the `new-constraint-name`.

ALTER TABLE...RENAME CONSTRAINT `constraint-name` TO `new-constraint-name`, when used for an RI constraint, only renames the constraint, not the underlying index or, if applicable, the foreign key role name. To rename the underlying index or the role name, use the ALTER INDEX statement.

table-alteration clauses

Use this clause to alter the following table attributes.

PCTFREE DEFAULT

Use this clause to change the percent free setting for the table to the default (200 bytes for a 4 KB, and up, page size).

[NOT] ENCRYPTED

Use this clause to change whether the table is encrypted. To encrypt a table, table encryption must already be enabled on the database. The table is encrypted using the encryption key and algorithm specified at database creation time.

After encrypting a table, any data for that table that was in temporary files or the transaction log before encryption still exists in unencrypted form. To address this, restart the database to remove the temporary files. Run the Backup utility (dbbackup) with the -o option, or use the BACKUP DATABASE statement, to back up the transaction log and start a new one.

When table encryption is enabled, table pages for the encrypted table, associated index pages, temporary file pages, and transaction log pages containing transactions on encrypted tables are encrypted.

DISABLE VIEW DEPENDENCIES clause

Use this clause to disable dependent regular views. Dependent materialized views are not disabled; you must disable each dependent materialized view by executing an ALTER MATERIALIZED VIEW...DISABLE statement.

ALTER OWNER clause

To alter the owner of a table:

- You must have the ALTER ANY OBJECT OWNER privilege.
- You must have one of the following privileges: ALTER privilege on the table, ALTER ANY TABLE privilege, or ALTER ANY OBJECT privilege.
- The new owner cannot already own a table with the same name.
- Enabled materialized views cannot reference the table.

PRESERVE or DROP PRIVILEGES

If you do not want the new owner to have the same privileges as the old owner, you can specify DROP PRIVILEGES (the default) to drop all explicitly granted privileges that allow a user access to the table. Implicitly granted privileges given to the owner of the table are given to the new owner and dropped from the old owner.

PRESERVE or DROP FOREIGN KEYS

To prevent the new owner from accessing data in referenced tables, you can specify DROP FOREIGN KEYS (the default) to drop all foreign keys within the table, as well as all foreign keys referring to the table.

Remarks

The ALTER TABLE statement changes table attributes (column definitions, constraints, and so on) in an existing table.

Avoid using more than one type of clause per statement.

The database server keeps track of object dependencies in the database. Alterations to the schema of a table may impact dependent views. Also, if there are materialized views that are dependent on the table you are attempting to alter, you must first disable them using the ALTER MATERIALIZED VIEW...DISABLE statement.

You cannot use ALTER TABLE on a local temporary table.

ALTER TABLE is prevented whenever the statement affects a table that is currently being used by another connection. ALTER TABLE can be time-consuming, and the database server does not process other requests referencing the table while the statement is being processed.

If you alter a column that a text index defined as IMMEDIATE REFRESH is built on, the text index is immediately rebuilt. If the text index is defined as AUTO REFRESH or MANUAL REFRESH, the text index is rebuilt the next time it is refreshed.

When you execute an ALTER TABLE statement, the database server attempts to restore column privileges on dependent views that are automatically recompiled. Privileges on columns that no longer exist in the recompiled views are lost.

ALTER TABLE requires exclusive access to the table.

Global temporary tables cannot be altered unless all users that have referenced the temporary table have disconnected.

This statement cannot be used within a snapshot transaction.

Privileges

You must be the owner of the table, or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege
- MANAGE ANY DBSPACE system privilege

To alter the table owner, you must also have the ALTER ANY OBJECT OWNER system privilege.

To create, alter, or drop indexes on a table, you must have the CREATE ANY INDEX, ALTER ANY INDEX, or DROP ANY INDEX system privilege, respectively.

Side Effects

Automatic commit.

A checkpoint is carried out at the beginning of the ALTER TABLE operation, and further checkpoints are suspended until the ALTER operation completes.

Once you alter a column or table, any stored procedures, views, or other items that refer to the altered column may no longer work.

If you change the declared length or type of a column, or drop a column, the statistics for that column are dropped.

Standards

ANSI/ISO SQL Standard

Core Feature. In the ANSI/ISO SQL Standard, ADD COLUMN and DROP COLUMN are supported as Core Features, as are ADD CONSTRAINT and DROP CONSTRAINT. ALTER [COLUMN] is SQL Feature F381, as is the ability to add, modify, or drop a DEFAULT value for a column. In the ANSI/ISO SQL Standard, altering the data type of a column is performed by specifying the SET DATA TYPE clause, which is SQL Language Feature F382. Conversely, the software supports modifying a column's data type through the ALTER clause directly.

Other clauses supported by the software, including ALTER CONSTRAINT, RENAME, PCTFREE, ENCRYPTED, and DISABLE MATERIALIZED VIEW, are not in the standard. Support for extensions to column definitions, and column and table constraint definitions, are either not in the standard, or are optional features of the standard.

Transact-SQL

ALTER TABLE is supported by Adaptive Server Enterprise. Adaptive Server Enterprise supports the ADD COLUMN and DROP COLUMN clauses, in addition to ADD CONSTRAINT and DROP CONSTRAINT. Adaptive Server Enterprise uses MODIFY rather than the keyword ALTER for the ALTER clause. Adaptive Server Enterprise uses the REPLACE clause for altering a column's DEFAULT value. In Adaptive Server Enterprise, ALTER TABLE is also used to enable/disable triggers for a specific table, a feature that is not supported in the software.

Example

The following example adds a new timestamp column, TimeStamp, to the Customers table. To run this next statement, you must have the ALTER privilege on the Customers table.

```
ALTER TABLE GROUPO.Customers
    ADD TimeStamp AS TIMESTAMP DEFAULT TIMESTAMP;
```

The following example drops the new timestamp column, TimeStamp that you added in the previous example.

```
ALTER TABLE GROUPO.Customers
    DROP TimeStamp;
```

Currently , the Street column in the Customers table can hold up to 30 characters. To allow it to hold up to 50 characters, execute the following:

```
ALTER TABLE GROUPO.Customers
    ALTER Street CHAR(50);
```

The following example adds a column to the Customers table, assigning each customer a sales contact. To run this next statement, you must also have the CREATE ANY INDEX system privilege because it creates a foreign key.

```
ALTER TABLE GROUPO.Customers
    ADD SalesContact INTEGER
    REFERENCES GROUPO.Employees ( EmployeeID )
    ON UPDATE CASCADE
    ON DELETE SET NULL;
```

This foreign key is constructed with cascading updates and is set to NULL on deletes. If an employee has their employee ID changed, the column is updated to reflect this change. If an employee leaves the company and has their employee ID deleted, the column is set to NULL.

The following example creates a foreign key, FK_SalesRepresentative_EmployeeID2, on the SalesOrders.SalesRepresentative column, linking it to Employees.EmployeeID. You must have the ALTER privilege on the SalesOrder table to execute the following statement:

```
ALTER TABLE GROUPO.SalesOrders
    ADD CONSTRAINT FK_SalesRepresentative_EmployeeID2
    FOREIGN KEY ( SalesRepresentative )
    REFERENCES GROUPO.Employees (EmployeeID);
```

The following example adds a column where the default is AUTOINCREMENT. In this example, all existing customer rows are modified to have a nullable AUTOINCREMENT column with the column value assigned, but the database server does not guarantee which row is assigned which value:

```
ALTER TABLE GROUPO.Customers
    ADD Surrogate_key INTEGER DEFAULT AUTOINCREMENT;
```

The following example changes the owner of the table mytable to Bob:

```
ALTER TABLE mytable
    ALTER OWNER TO bob;
```

The following example creates the table library_books and then adds a new column using ALTER TABLE:

```
CREATE OR REPLACE TABLE library_books (
```

```
    isbn CHAR(20) CONSTRAINT cons1 PRIMARY KEY NONCLUSTERED,
    title CHAR(100) NOT NULL,
    author CHAR(50),
    copyright_date DATE
);
ALTER TABLE library_books ADD dewey AS CHAR(20) NULL
    CONSTRAINT cons2 UNIQUE CLUSTERED
        CHECK ( SUBSTRING(dewey,1,3) >= '000' AND SUBSTRING(dewey,1,3) <=
'999' );
```

Related Information

Strings [page 11]
Events
Reloading Tables with AUTOINCREMENT Columns
Table Alteration
The Special IDENTITY Column
Special Values [page 87]
Table and Column Constraints
Use of a Sequence to Generate Unique Values
Clustered Indexes
View Dependencies
%TYPE and %ROWTYPE Attributes [page 115]
Encrypting a Table
sa_reset_identity System Procedure [page 1688]
ALTER MATERIALIZED VIEW Statement [page 692]
BACKUP DATABASE Statement [page 778]
ALTER INDEX Statement [page 685]
CREATE TABLE Statement [page 1002]
DROP TABLE Statement [page 1134]
Backup Utility (dbbackup)
SQL Data Types [page 129]
allow_nulls_by_default Option
SQL Variables [page 123]
@@identity Global Variable [page 126]

1.4.4.28 ALTER TEXT CONFIGURATION Statement

Alters a text configuration object.

Syntax

```
ALTER TEXT CONFIGURATION [ owner.]config-name
STOPLIST stoplist-string
| DROP STOPLIST
| { MINIMUM | MAXIMUM } TERM LENGTH integer }
```

```
| TERM BREAKER { GENERIC [ EXTERNAL NAME external-call ] | NGRAM }
| PREFILTER EXTERNAL NAME external-call
| DROP PREFILTER
| SAVE OPTION VALUES [ FROM CONNECTION ]

external-call :
[ system-configuration:]function-name@library-file-prefix[.{ so | dll} ]

system-configuration :
{ generic-operating-system | specific-operating-system } [ (processor-
architecture) ]

generic-operating-system :
{ UNIX | Windows }

specific-operating-system :
{ AIX | HPUX | Linux | OSX | Solaris | WindowsNT }

processor-architecture :
{ 32 | 64 | ARM | IA64 | PPC | SPARC | X86 | X86_64 }
```

Parameters

STOPLIST clause

Use this clause to create or replace the list of terms to ignore when building a text index. Using this text configuration object, terms specified in this list are also ignored in a query. Separate stoplist terms with spaces. For example, `STOPLIST 'because about therefore only'`. Stoplist terms cannot contain whitespace.

Samples of stoplists for different languages are located in *%SQLANYSAMP17%*`\SQLAnywhere\SQL`.

Stoplist terms should not contain non-alphanumeric characters. The stoplist length must be less than 8000 bytes.

Carefully consider whether you want to put terms in your stoplist.

DROP STOPLIST clause

Use this clause to drop the stoplist for a text configuration object.

MINIMUM TERM LENGTH clause

MINIMUM TERM LENGTH clause is ignored when using NGRAM text indexes.

The minimum length, in characters, of a term to include in the text index. Terms that are shorter than this setting are ignored when building or refreshing the text index. The value of this option must be greater than 0. If you set this option to be higher than MAXIMUM TERM LENGTH, the value of MAXIMUM TERM LENGTH is automatically adjusted to be the same as the new MINIMUM TERM LENGTH value.

MAXIMUM TERM LENGTH clause

With NGRAM text indexes, use the MAXIMUM TERM LENGTH clause to set the size of the n-grams into which strings are broken. Terms shorter than MAXIMUM TERM LENGTH are not indexed.

With GENERIC text indexes, use the MAXIMUM TERM LENGTH clause to set the maximum length, in characters, of a term to include in the text index. Terms that are longer than this setting are ignored when building or refreshing the text index. The value of MAXIMUM TERM LENGTH must be less than or equal to 60. If you set this option to be lower than MINIMUM TERM LENGTH, the value of MINIMUM TERM LENGTH is automatically adjusted to be the same as the new MAXIMUM TERM LENGTH value.

TERM BREAKER clause

The name of the algorithm to use for separating column values into terms. The choices are GENERIC (the default) or NGRAM.

GENERIC

For GENERIC, you can use the built-in GENERIC term breaker algorithm by specifying TERM BREAKER GENERIC, or you can specify an external algorithm using the TERM BREAKER GENERIC EXTERNAL NAME clause.

The built-in GENERIC algorithm treats any string of one or more alphanumerics, separated by non-alphanumerics, as a term.

Specify the TERM BREAKER GENERIC EXTERNAL NAME clause to specify an entry point to a term breaker function in an external library. This is useful if you have custom requirements for how you want terms broken up before they are indexed or queried (for example, if you want an apostrophe to be considered as part of a term and not as a term breaker).

`external-call` can specify more than one function and/or library, and can include the file extension of the library, which is typically `.dll` on Windows, and `.so` on UNIX and Linux. In the absence of the file extension, the database server defaults to the platform-specific file extension for libraries. For example, `EXTERNAL NAME 'TermBreakFunct1@myTBlib;unix:TermBreakFunct2@myTBlib'` calls the TermBreakFunct1 function from `myTBlib.dll` on Windows, and the TermBreakFunct2 function from `myTBlib.so` on UNIX or Linux.

NGRAM

The built-in NGRAM algorithm breaks strings into n-grams. An n-gram is an `n`-character substring of a larger string. The NGRAM term breaker is required for fuzzy (approximate) matching, or for documents that do not use whitespace or non-alphanumeric characters to separate terms, if no external term breaker is specified.

PREFILTER EXTERNAL NAME clause

Specify the PREFILTER EXTERNAL NAME clause to specify an entry point to a prefilter function in an external library. This is useful if text data needs to be extracted from binary data (for example, PDF). It is also useful if the text you want to index contains formatting information and/or images that you want to strip out before indexing the data (for example, HTML).

Because `external-call` can specify multiple sets of operating systems, processors, libraries, and functions, more-precisely specified configurations take precedence over less-precisely defined configurations. For example, `Solaris(X86_64):myfunc64@mylib.so` takes precedence over `Solaris:myfunc64@mylib.so`, and so on.

For syntaxes that support `system-configuration`, if you do not specify `system-configuration`, then it is assumed that the procedure runs on all system configurations. *UNIX* represents the following operating systems: AIX, HP-UX, Linux, macOS, and Solaris. The generic term *Windows* represents all versions of the Windows operating system.

If you specify *UNIX* for one of the calls, then it is assumed that the other call is for Windows.

The `specific-operating-system` and `processor-architecture` values are those operating systems and processors supported by SQL Anywhere server.

The library name (`library-file-prefix`) is followed by the file extension, which is typically `.dll` on Windows and `.so` on UNIX and Linux. In the absence of the file extension, the database server defaults to the platform-specific file extension for libraries. For example, `PREFILTER EXTERNAL NAME 'PrefilterFunct1@myPreFilterlib;unix:PrefilterFunct2@myPreFilterlib'` calls the PrefilterFunct1 function from `myPreFilterlib.dll` on Windows, and the PrefilterFunct2 function from `myPreFilterlib.so` on UNIX and Linux.

DROP PREFILTER clause

Use the DROP PREFILTER clause to drop use of the specified prefiltering library for the text configuration object. Prefiltering is no longer performed when the database server builds indexes that use this text configuration object.

SAVE OPTION VALUES clause

When a text configuration object is created, the current date_format, time_format, timestamp_format, and timestamp_with_time_zone_format database options reflect how DATE, TIME, and TIMESTAMP columns are saved with the text configuration object. Use the SAVE OPTION VALUES clause to update the option values saved for the text configuration object to reflect the options currently in effect for the connection.

Remarks

Before changing the term length settings, read about the impact of various settings on what gets indexed and how query terms are interpreted.

Text indexes are dependent on a text configuration object. Before using this statement you must truncate dependent AUTO or MANUAL REFRESH text indexes, and drop any IMMEDIATE REFRESH text indexes.

To view the settings for text configuration objects, query the SYSTEXTCONFIG system view.

Privileges

You must be the owner of the text configuration object, or have one of the following privileges:

- ALTER privilege on the text configuration object
- ALTER ANY TEXT CONFIGURATION system privilege
- ALTER ANY OBJECT system privilege

If you are altering an external term breaker, you must also have the CREATE EXTERNAL REFERENCE system privilege.

When specifying or dropping a prefilter, or specifying an external term breaker, you must *also* have the ALTER ANY TEXT CONFIGURATION or ALTER ANY OBJECT system privilege.

Side Effects

Automatic commit

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements create a text configuration object, myTextConfig, and then change the maximum term length to 16. To run this statement, you'll need the CREATE TEXT CONFIGURATION system privilege:

```
CREATE TEXT CONFIGURATION myTextConfig FROM default_char;
ALTER TEXT CONFIGURATION maxTerm16
   MAXIMUM TERM LENGTH 16;
```

The following statement adds a stoplist to the myTextConfig configuration object:

```
ALTER TEXT CONFIGURATION myTextConfig
   STOPLIST 'because about therefore only';
```

The following statement configures an external term breaker for the myTextConfig text configuration object. Both the *Windows* and *UNIX* interfaces are specified.

```
ALTER TEXT CONFIGURATION myTextConfig
   TERM BREAKER GENERIC
   EXTERNAL NAME
'my_termbreaker@termbreaker.dll;unix:my_termbreaker@libtermbreaker_r.so'
```

The following example configures an external prefilter for the myTextConfig text configuration object. Both the *Windows* and *UNIX* interfaces are specified.

```
ALTER TEXT CONFIGURATION myTextConfig
   PREFILTER EXTERNAL NAME
'html_xml_filter@html_xml_filter.dll;unix:html_xml_filter@libhtml_xml_filter_r.so
';
```

The following example drops the external prefilter for the myTextConfig text configuration object.

```
ALTER TEXT CONFIGURATION myTextConfig DROP PREFILTER;
```

Related Information

What to Specify When Creating or Altering Text Configuration Objects

Altering a Text Configuration Object
Viewing Text Index Terms and Settings (SQL Central)
Tutorial: Performing a Full Text Search on a GENERIC TeIt index
Tutorial: Performing a Fuzzy Full Text Search
CREATE TEXT CONFIGURATION Statement [page 1028]
DROP TEXT CONFIGURATION Statement [page 1135]
sa_char_terms System Procedure [page 1534]
sa_nchar_terms System Procedure [page 1664]
sa_refresh_text_indexes System Procedure [page 1683]
sa_text_index_stats System Procedure [page 1735]
SYSTEXTCONFIG System View [page 1958]

1.4.4.29 ALTER TEXT INDEX Statement

Alters the definition of a text index.

Syntax

```
ALTER TEXT INDEX [ owner.]text-index-name
ON [ owner.]table-name
alter-clause
```

```
alter-clause :
rename-object
| refresh-alteration
```

```
rename-object :
 RENAME { AS | TO } new-name
```

```
refresh-alteration :
{ MANUAL REFRESH
| AUTO REFRESH [ EVERY integer { MINUTES | HOURS } ] }
```

Parameters

RENAME clause

Use the RENAME clause to rename the text index.

REFRESH clause

Specify the REFRESH clause to set the refresh type for the text index.

Remarks

Once a text index is created, you cannot change it to, or from, IMMEDIATE REFRESH. If either of these changes is required, you must drop and recreate the text index.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

You can only alter a text index built on a materialized view by renaming it. You cannot change the refresh type for a text index built on a materialized view.

Privileges

To alter a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

To alter a text index on a materialized view, you must be the owner of the materialized, or have one of the following privileges:

- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

Side Effects

Automatic commit

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The first statement creates a text index, txt_index_manual, defining it as MANUAL REFRESH. The second statement alters the text index to refresh automatically every day. The third statement renames the text index to txt_index_daily.

```
CREATE TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation ( Description )
```

```
   MANUAL REFRESH;
ALTER TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation
   AUTO REFRESH EVERY 24 HOURS;
ALTER TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation
   RENAME AS txt_index_daily;
```

Related Information

Snapshot Isolation
Full Text Search
Viewing Text Index Terms and Settings (SQL Central)
Tutorial: Performing a Full Text Search on a GENERIC TeIt index
Tutorial: Performing a Fuzzy Full Text Search
CREATE TEXT INDEX Statement [page 1030]
DROP TEXT INDEX Statement [page 1136]
REFRESH TEXT INDEX Statement [page 1325]
TRUNCATE TEXT INDEX Statement [page 1445]
COMMENT Statement [page 808]

1.4.4.30 ALTER TIME ZONE Statement

Modifies a time zone object.

Syntax

```
ALTER TIME ZONE name time-zone-option [ ... ]
```

```
time-zone-option :
  { OFFSET offset
  | DST OFFSET offset
  | NO DST
  | [ DST ] STARTING month-day-rule AT { minutes | hours:minutes }
  | [ DST ] ENDING month-day-rule AT { minutes | hours:minutes } }
```

Remarks

All clauses are optional; however, if none are present, then the statement fails.

If you are adding daylight savings time to a time zone that previously had no daylight savings time, then use the STARTING and ENDING clauses. If you are removing daylight savings time from a time zone, then specify the NO DST clause and omit the STARTING and ENDING clauses.

Privileges

You must have the MANAGE TIME ZONE system privilege or the ALTER ANY OBJECT system privilege.

Side Effects

Automatic commit.

Executing this statement populates the ISYSTIMEZONE system table.

Example

To change the time zone named EasternTime, which uses daylight savings time, to use Australian Eastern Time without daylight savings time, execute the following statement:

```
ALTER TIME ZONE EasternTime OFFSET '10:00'
NO DST;
```

Related Information

COMMENT Statement [page 808]
CREATE TIME ZONE Statement [page 1033]
DROP TIME ZONE Statement [page 1138]
time_zone Option
SYSTIMEZONE System View [page 1961]

1.4.4.31 ALTER TRACE EVENT SESSION Statement

Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

Syntax

```
ALTER TRACE EVENT SESSION session-name
[ ON SERVER ]
{ ADD TRACE EVENT trace-event-name [,...]
  | DROP TRACE EVENT trace-event-name [,...]
  | ADD TARGET FILE [ ( SET target-parameter-name=target-parameter-value
[, ...] ) ]
  | DROP TARGET target-name [, ...] ]
  | STATE = { START | STOP }
```

```
}

target-parameter-name :
{ filename_prefix
  | max_size
  | num_files
  | flush_on_write
  | compressed }
```

Parameters

ON SERVER clause

Alters a trace event session that is recording trace events from all databases on the database server. If this clause is not specified, then only the trace event session on the current database is altered.

ADD TRACE EVENT

The name of the trace event being added to the session.

DROP TRACE EVENT

The name of the trace event being removed from the session.

ADD TARGET

The name of the target being added to the session. Information about the trace events that are part of the session is logged to this target (file).

DROP TARGET

The name of the target being removed from the session.

Remarks

Adding or dropping trace events or targets from a running trace session causes the session to pause briefly to make the changes and then resume after the changes are made. You do not need to stop a tracing session to add or drop trace events or targets. Adding or removing a trace event from a session that has already started has the side effect that some trace events are missed while a session is temporarily paused.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a trace event called my_event, creates a trace event session called my_session, and starts the session:

```
CREATE TEMPORARY TRACE EVENT my_event( id INTEGER, information LONG VARCHAR );
CREATE TEMPORARY TRACE EVENT SESSION my_session
    ADD TRACE EVENT my_event, -- user event
    ADD TRACE EVENT SYS_ConsoleLog_Information -- system event
    ADD TARGET FILE ( SET filename_prefix='my_trace_file' ); -- add a target
ALTER TRACE EVENT SESSION my_session
     STATE = START;
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.4.4.32 ALTER TRIGGER Statement

Replaces a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.

Syntax

Change the definition of a trigger

```
ALTER TRIGGER trigger-name trigger-definition
```

```
trigger-definition : CREATE TRIGGER syntax
```

Obfuscate a trigger definition

```
ALTER TRIGGER trigger-name ON [owner.] table-name SET HIDDEN
```

Remarks

Change the definition of a trigger

The ALTER TRIGGER statement is identical in syntax to the CREATE TRIGGER statement except for the first word.

Either the Transact-SQL or Watcom SQL form of the CREATE TRIGGER syntax can be used.

Obfuscate a trigger definition

You can use SET HIDDEN to obfuscate the definition of the associated trigger and cause it to become unreadable. The trigger can be unloaded and reloaded into other databases. If SET HIDDEN is used, debugging using the debugger does not show the trigger definition, nor is it available through procedure profiling.

> **Note**
>
> The SET HIDDEN operation is irreversible.

Privileges

You must be the owner of the underlying table, or have the one of the following privileges:

- ALTER privilege on the underlying table with the CREATE ANY OBJECT system privilege
- ALTER ANY TRIGGER system privilege
- ALTER ANY OBJECT system privilege

To alter a trigger on a view owned by someone else, you must have either the ALTER ANY TRIGGER and ALTER ANY VIEW system privileges, or you must have the ALTER ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Hiding the Contents of a Procedure, Function, Trigger, Event, or View
CREATE TRIGGER Statement [page 1036]
CREATE TRIGGER Statement [T-SQL] [page 1043]
DROP TRIGGER Statement [page 1142]

1.4.4.33 ALTER USER Statement

Alters user settings.

Syntax

Change the definition of a database user

```
ALTER USER user-name
  [ IDENTIFIED BY password ]
  [ LOGIN POLICY policy-name ]
  [ FORCE PASSWORD CHANGE { ON | OFF } ]
```

Unlock a database user

```
ALTER USER user-name
[ RESET LOGIN POLICY ]
```

Refresh the Distinguished Name (DN) for an LDAP user

```
ALTER USER user-name
REFRESH DN
```

Change a password part

```
ALTER USER user-name
  [ IDENTIFIED { FIRST | LAST } BY password-part ]
```

Parameters

user-name

The name of the user.

IDENTIFIED clause

The password of the user. A user without a password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID.

IDENTIFIED BY clause

Use this clause to reset the password for a user. To reset a user to not have a password, set `password` to NULL.

IDENTIFIED { FIRST | LAST } BY clause

Use this clause to reset part of the password for a user who has a dual control password. A user with dual control password has the CHANGE_PASSWORD_DUAL_CONTROL login policy option enabled for their login policy.

policy-name

The name of the login policy to assign the user. No change is made if the LOGIN POLICY clause is not specified.

FORCE PASSWORD CHANGE clause

Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the user's policy.

RESET LOGIN POLICY clause

Resets the number of failed login attempts, as well as the user's last login time and last failed login time. If a user account was locked for exceeding login policy limits, it is unlocked.

REFRESH DN clause

REFRESH DN clears the Distinguished Name (DN) and timestamp of the user so that at the time of the next LDAP authentication, the search for the DN is done. If the authentication succeeds during the next LDAP authentication of this user then both the DN and the timestamp are updated with the new DN and current time.

Remarks

Two administrators are required to reset a dual control password. One administrator executes the IDENTIFIED FIRST BY clause to set the first part of the password and another administer executes the IDENTIFIED LAST BY clause to set the last part of the password. The user combines the two password parts and uses this combined password to connect to the database.

The requirements and restrictions for a dual-control password part are the same as those described for a password except that the maximum length of each part is 127 bytes.

The verify_password_function login policy option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement.

If you use one of the LOGIN POLICY or RESET LOGIN POLICY clauses and if the password_expiry_on_next_login value is set to ON in the user's assigned login policy, then the user's password expires immediately when they next login even if they are assigned to the same policy. If the password_expiry_on_next_login value is set to OFF in the user's assigned login policy, then you can use the FORCE PASSWORD CHANGE clause to force the user to change their password when they next login.

The IDENTIFIED BY and REFRESH DN clauses alone do not affect password expiry, even if the password_expiry_on_next_login value is set to ON in the user's assigned login policy. If you are setting a temporary password for the user, then use the FORCE PASSWORD CHANGE clause to force the user to change their password when they next login.

The ALTER USER...REFRESH DN syntax clears the Distinguished Name (DN) and timestamp of a user so that during the next LDAP authentication, a search for the DN is performed, instead of using the cached DN, which can become out of date. If the authentication succeeds, then both the DN and the timestamp are updated with the new DN and current time.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

Any user can change their own password.

To change passwords for other users, you must have the CHANGE PASSWORD system privilege.

For all other changes to other users, including forcing users to change their password, you must have the MANAGE ANY USER system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement alters a user named SQLTester, setting their password to welcome123, setting their login policy to Test1, and allowing them to bypass a forced password change.

```
ALTER USER SQLTester IDENTIFIED BY welcome123
LOGIN POLICY Test1
FORCE PASSWORD CHANGE OFF;
```

The following example refreshes the LDAP Distinguished Name for user myusername.

```
ALTER USER myusername REFRESH DN;
```

Related Information

Password and User ID Restrictions and Considerations
Dual Control Passwords
Login Policies
Password and User ID Restrictions and Considerations
Assigning a Login Policy to an Existing User
ALTER LOGIN POLICY Statement [page 690]
COMMENT Statement [page 808]
CREATE LOGIN POLICY Statement [page 894]
CREATE USER Statement [page 1045]
DROP LOGIN POLICY Statement [page 1101]
DROP USER Statement [page 1143]
verify_password_function Option
GRANT CONNECT Statement [page 1197]

1.4.4.34 ALTER VIEW Statement

Replaces a view definition with a modified version.

Syntax

Change the definition of a view

```
ALTER VIEW
[ owner.]view-name [ ( column-name, ... ) ] AS query-expression
[ WITH CHECK OPTION ]
```

Change the attributes of a view

```
ALTER VIEW
[ owner.]view-name { SET HIDDEN | RECOMPILE | DISABLE | ENABLE }
```

Parameters

AS clause

The SELECT statement on which the view is based. The SELECT statement must not refer to local temporary tables. Also, `query-expression` can have a GROUP BY, HAVING, WINDOW, or ORDER BY clause, and can contain UNION, EXCEPT, INTERSECT, or a common table expression.

Query semantics dictate that the order of the rows returned is undefined unless the query combines an ORDER BY clause with a TOP or FIRST clause in the SELECT statement.

WITH CHECK OPTION clause

The WITH CHECK OPTION clause rejects any updates and inserts to the view that do not meet the criteria of the view as defined by its `query-expression`.

SET HIDDEN clause

Use the SET HIDDEN clause to obfuscate the definition of the view and cause the view to become hidden from view, for example in *SQL Central*. Explicit references to the view still work.

> **i** Note
>
> The SET HIDDEN operation is irreversible.

RECOMPILE clause

Use the RECOMPILE clause to re-create the column definitions for the view. This clause is identical in functionality to the ENABLE clause, except that it can be used on a view that is not disabled. When a view is recompiled, the database server restores the column privileges based on the column names specified in the new view definition. The existing privileges are lost when a column no longer exists after the recompilation.

DISABLE clause

Use the DISABLE clause to disable the view from use by the database server.

ENABLE clause

Use the ENABLE clause to enable a disabled view. Enabling the view causes the database server to re-create the column definitions for the view. Before you enable a view, you must enable any views upon which it depends.

Remarks

The `query-expression` can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

If you execute an ALTER VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. You must drop the trigger before the view can be dropped or altered.

If you alter a view owned by another user, you must qualify the name by including the owner (for example, GROUPO.ViewSalesOrders). If you don't qualify the name, the database server looks for a view with that name owned by you and alters it. If there isn't one, it returns an error.

When you alter a view, existing privileges on the view are maintained, and do not have to be reassigned. Instead of using the ALTER VIEW statement, you could also drop the view and recreate it using the DROP VIEW and CREATE VIEW, respectively. However, if you do so, privileges on the view need to be reassigned.

A query can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

After completing the view alteration using the syntax to change the definition of a view, the database server recompiles the view. Depending on the type of change you made, if there are dependent views, the database server attempts to recompile them as well. If you have made a change that impacts a dependent view, you may need to alter the definition for the dependent view as well.

> ⚠ Caution
>
> If the SELECT statement defining the view contained an asterisk (*), the number of the columns in the view may change if columns have been added or deleted from the underlying tables. The names and data types of the view columns may also change.

Change the definition of a view

This syntax is used to alter the structure of the view. Unlike altering tables where your change may be limited to individual columns, altering the structure of a view requires you to replace the entire view definition with a new definition, much as you would for creating the view.

Change the attributes of a view

This syntax is used to change attributes for the view, such as whether the view definition is hidden.

When you use SET HIDDEN, the view can be unloaded and reloaded into other databases. If SET HIDDEN is used, debugging using the debugger does not show the view definition, nor is it be available through procedure profiling. If you need to change the definition of a hidden view, you must drop the view and create it again using the CREATE VIEW statement.

When you use the DISABLE clause, the view is no longer available for use by the database server for answering queries. Disabling a view is similar to dropping it, except that the view definition remains in the database. Disabling a view also disables any dependent views. Therefore, the DISABLE clause requires exclusive access not only to the view being disabled, but also any dependent views, since they are disabled too.

Privileges

You must be the owner of the view, or have one of the following privileges:

- ALTER ANY VIEW system privilege
- ALTER ANY OBJECT system privilege

You must also have permission to select from the underlying objects for the view.

Side Effects

Automatic commit.

All procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the new view definition. The unloading and loading of procedures and triggers can have a performance impact if you are regularly altering views.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

View Dependencies
Views
Hiding the Contents of a Procedure, Function, Trigger, Event, or View
Creating a Regular View
CREATE VIEW Statement [page 1051]
DROP VIEW Statement [page 1147]
CREATE MATERIALIZED VIEW Statement [page 896]
ALTER MATERIALIZED VIEW Statement [page 692]

1.4.4.35 ATTACH TRACING Statement (Deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The ATTACH TRACING statement starts a diagnostic tracing session (starts sending diagnostic information to the diagnostic tables).

Syntax

```
ATTACH TRACING TO { LOCAL DATABASE | connect-string }
  [ LIMIT { size | history } ]
```

```
connect-string : the connection string for the database
```

```
size : SIZE nnn { MB | GB }
```

```
history : HISTORY nnn { MINUTES | HOURS | DAYS }

nnn : integer
```

Parameters

connect-string

The connection string is required to connect to the database receiving the tracing information. This parameter is only required when the database being profiled is different from the database receiving the data.

The following connection parameters are allowed in `connect-string`: DBF, DBKEY, DBN, Host, Server, LINKS, PWD, UID.

Specify the DBF relative to the database server that you want to connect to. If you do not specify a different database server, then the database server to which you are currently connected attempts to start the tracing database identified by the DBF connection parameter.

An error is returned if you specify the DBF parameter with the LINKS or Server connection parameters.

LIMIT clause

The volume limit of data stored in the tracing database, either by size, or by length of time.

Remarks

The ATTACH TRACING statement is used to start a tracing session for the database you want to profile. You can only use it once a tracing level has been set. You can set the tracing level using the sa_set_tracing_level system procedure.

Once a session is started, tracing information is generated according to the tracing levels set in the sa_diagnostic_tracing_level table. You can send the tracing data to tracing tables within the same database that is being profiled by specifying LOCAL DATABASE. Alternatively, you can send the tracing data to a separate tracing database by specifying a connection string (`connect-string`) to that database. The tracing database must already exist, and you must have permissions to access it.

You can limit the amount of tracing data to store using the LIMIT SIZE or LIMIT HISTORY clauses. Use the LIMIT SIZE clause when you want to limit the volume of tracing data to a certain size, as measured in megabytes or gigabytes. Use the LIMIT HISTORY clause to limit the volume of tracing data to a period of time, as measured in minutes, hours, or days. For example, `HISTORY 8 DAYS` limits the amount of tracing data stored in the tracing database to 8 days' worth.

To start a tracing session, TCP/IP must be running on the database server(s) on which the tracing database and production database are running.

Packets that contain potentially sensitive data are visible on the network interface, even when tracing to a local database. For security purposes, you can specify encryption in the connection string.

To see the current tracing levels set for a database, look in the sa_diagnostic_tracing_level table.

To see where tracing data is being sent to, examine the SendingTracingTo database property.

You must be connected to the database being profiled to execute this statement.

Privileges

You must have the DIAGNOSTICS system role and the MANAGE PROFILING system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example sets the tracing level to 1 using the sa_set_tracing_level system procedure. Then it starts a tracing session. Tracing data generated for the local database will be sent to the mytracingdb tracing database on another computer, as shown by the specified connection string. A maximum of two hours of tracing data will be maintained during the tracing session.

```
CALL sa_set_tracing_level( 1 );
ATTACH TRACING TO
'UID=DBA;PWD=passwd;Server=remotedbsrv;DBN=mytracingdb;Host=mytracing-pc'
  LIMIT HISTORY 2 HOURS;
```

Related Information

SQL Anywhere Profiler
Alphabetical List of Connection Parameters
TCP/IP Protocol
Diagnostic Tracing (Deprecated)
DETACH TRACING Statement (Deprecated) [page 1084]
REFRESH TRACING LEVEL Statement (Deprecated) [page 1327]
-x Database Server Option

sa_diagnostic_tracing_level Table (Deprecated) [page 1508]
sa_set_tracing_level System Procedure (Deprecated) [page 1719]

1.4.4.36 BACKUP DATABASE Statement

Backs up a database and transaction log.

Syntax

Image backup

```
BACKUP DATABASE
DIRECTORY backup-directory
[ backup-option [  backup-option ... ] ]
```

```
backup-directory : { string | variable }
```

```
backup-option :
WAIT BEFORE START
| WAIT AFTER END
| DBFILE ONLY
| TRANSACTION LOG ONLY
| TRANSACTION LOG RENAME [ MATCH ]
| TRANSACTION LOG TRUNCATE
| ON EXISTING ERROR
| WITH COMMENT comment-string
| HISTORY { ON | OFF }
| AUTO TUNE WRITERS { ON | OFF }
| WITH CHECKPOINT LOG { AUTO | COPY | NO COPY | RECOVER }
```

Archive backup

```
BACKUP DATABASE TO archive-root
[ backup-option [  backup-option ... ] ]
```

```
archive-root :  { string | variable }
```

```
backup-option :
WAIT BEFORE START
| WAIT AFTER END
| DBFILE ONLY
| TRANSACTION LOG ONLY
| TRANSACTION LOG RENAME [ MATCH ]
| TRANSACTION LOG TRUNCATE
| ATTENDED { ON | OFF }
| WITH COMMENT comment-string
| HISTORY { ON | OFF }
| WITH CHECKPOINT LOG [ NO ] COPY
| MAX WRITE { number-of-writers | AUTO }
| FREE PAGE ELIMINATION { ON | OFF }
```

```
comment-string : string
```

```
number-of-writers : integer
```

Parameters

DIRECTORY clause

The target location on disk for the backup files, relative to the database server's current directory at startup. If the directory does not exist, it is created. Specifying an empty string as a directory allows you to rename or truncate the transaction log without first making a copy of it. Do not use this clause if you are using database mirroring.

WAIT BEFORE START clause

This clause delays the backup until there are no active transactions. All other activity on the database is prevented and a checkpoint is performed.

Using this clause with the WITH CHECKPOINT LOG NO COPY clause verifies that the backup copy of the database does not require recovery and allows you to start the backup copy of the database in read-only mode and validate it. When you validate the backup database, you do not need to make an additional copy of the database.

WAIT AFTER END clause

This clause ensures that all transactions are completed before the transaction log is renamed or truncated. The database server waits for other connections to commit or rollback any open transactions before finishing the backup. Use this clause with caution as new, incoming transactions can cause the backup to wait indefinitely.

DBFILE ONLY clause

This clause makes backup copies of the main database file and all associated dbspaces, but not the transaction log. You cannot use the DBFILE ONLY clause with the TRANSACTION LOG RENAME or TRANSACTION LOG TRUNCATE clauses.

TRANSACTION LOG ONLY clause

You can specify the TRANSACTION LOG ONLY clause to create a backup copy of the transaction log, without copying the other database files.

TRANSACTION LOG RENAME [MATCH] clause

This clause causes the database server to rename the current transaction log to a file name of the form `YYMMDDaa.log` and start a *new* transaction log that has the same name as the database file. The backup transaction log gets the same name as the active transaction log unless MATCH is specified. If MATCH is specified, the backup copy of the transaction log gets the same name as the renamed file (`YYMMDDaa.log`). Using the MATCH keyword enables the same statement to be executed several times without writing over old data. The filename `YYMMDDaa.log` begins with the current year, month, day and the uppercase letters AA. This is followed by AB, AC, and so on.

The transaction log can be renamed and restarted without completing a backup by specifying an empty directory name with the TRANSACTION LOG ONLY clause. For example:

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

TRANSACTION LOG TRUNCATE clause

If this clause is used, the current transaction log is truncated and restarted at the completion of the backup. Do not use this clause if you are using database mirroring.

The transaction log can be truncated without completing a backup by specifying an empty directory name with the TRANSACTION LOG ONLY clause. For example:

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG TRUNCATE;
```

ON EXISTING ERROR clause

This clause applies only to image backups. By default, existing files are overwritten when you execute a BACKUP DATABASE statement. If this clause is used, an error occurs if any of the files to be created by the backup already exist.

WITH COMMENT clause

This clause records a comment in the backup history file. For archive backups, the comment is also recorded in the archive file.

HISTORY clause

This clause enables or disables backup history. By default, this clause is ON, meaning that each backup operation appends a line to the `backup.syb` file. Specifying HISTORY OFF prevents updates to the `backup.syb` file, and is recommended when:

- The database is backed up frequently.
- There is no procedure in place to periodically archive or delete the `backup.syb` file.
- Disk space is limited.

AUTO TUNE WRITERS clause

Specifying this clause enables or disables the automatic tuning of writers. During the backup process, one writer writes the backup files to the backup directory. If the backup directory is on a device that can handle an increased writer load (such as a RAID array), the default AUTO TUNE WRITERS ON improves overall backup performance by increasing the number of writers. The database server periodically examines the read and write performances of all devices that are participating in the backup. Specifying AUTO TUNE WRITERS OFF prevents the database server from creating additional writers.

WITH CHECKPOINT LOG clause

This clause specifies how the backup processes the database files before writing them to the destination directory. The choice of whether to apply pre-images during a backup, or copy the checkpoint log as part of the backup, has performance implications. The default setting is AUTO for image backups and COPY for archive backups.

COPY clause

This option cannot be used with the WAIT BEFORE START clause of the BACKUP DATABASE statement.

When you specify COPY, the backup reads the database files without applying any modified pages. The entire checkpoint log and the system dbspace are copied to the backup directory. The next time the database server is started, the database server automatically recovers the database to the state it was in as of the checkpoint at the time the backup started.

Because pages do not have to be written to the temporary file, using this option can provide better backup performance, and reduce internal server contention for other connections that are operating during a backup. However, since the checkpoint log contains original images of modified pages, it grows in the presence of database updates. With copy specified, the backed-up copy of the database

files may be larger than the database files at the time the backup started. The COPY option should be used if disk space in the destination directory is not an issue.

NO COPY clause

When you specify NO COPY, the checkpoint log is not copied as part of the backup. This option causes modified pages to be saved in the temporary file so that they can be applied to the backup as it progresses. The backup copies of the database files are the same size as the database when the backup operation commenced.

This option results in smaller backed up database files, but the backup may proceed more slowly, and possibly decrease performance of other operations in the database server. It is useful in situations where space on the destination drive is limited.

RECOVER clause

When you specify RECOVER, the database server copies the checkpoint log (as with the COPY option), but applies the checkpoint log to the database when the backup is complete. This restores the backed up database files to the same state (and size) that they were in at the start of the backup operation. This option is useful if space on the backup drive is limited (it requires the same amount of space as the COPY option for backing up the checkpoint log, but the resulting file size is smaller).

AUTO clause

When you specify AUTO, the database server checks the amount of available disk space on the volume hosting the backup directory. If there is at least twice as much disk space available as the size of the database at the start of the backup, then this option behaves as if copy were specified. Otherwise, it behaves as NO COPY. AUTO is the default behavior.

MAX WRITE clause

For archive backups, by default one thread is dedicated to writing the backup files. If the backup directory is on a device that can handle an increased writer load (such as a RAID array), then overall backup performance can be improved by increasing the number of threads acting as writers.

If AUTO is specified, one output stream is created for each reader thread. The value `n` specifies the maximum number of output streams that can be created, up to the number of reader threads. The default value for this clause is 1.

The first stream, stream 0, produces files named `myarchive.X`, where `X` is a number that starts at 1 and continues incrementing to the number of files required. All of the other streams produce files named `myarchive.Y.Z`, where `Y` is the stream number (starting at 1), and `Z` is a number that starts at 1 and continues incrementing to the number of files required.

FREE PAGE ELIMINATION clause

By default, archive backups skip some free pages, which can result in smaller and potentially faster backups. Free page elimination has no effect on the back up of transaction log files because transaction log files do not contain free pages. Databases with large transaction log files may not benefit as much from free page elimination as databases with small transaction log files.

When you back up a strongly encrypted database with free page elimination turned on, you must specify the encryption key when restoring the database. When you back up a strongly encrypted database with free page elimination turned off, you do not need to specify the encryption key when restoring the database.

As of version 12, you cannot restore archive backups created with version 11 or earlier database servers.

Remarks

The BACKUP DATABASE statement performs a server-side backup. To perform a client-side backup, use the dbbackup utility.

If the disk sandbox feature is enabled for the database, you must specify a secure feature key that disables the disk sandbox feature for the database server to be able to make the backup in a directory outside of the sandbox (the directory where the main database file is located and any subdirectories of this directory).

Each backup operation, whether image or archive, updates a history file called `backup.syb`. This file records the BACKUP and RESTORE operations that have been performed on a database server.

To create a backup that can be started on a read-only server without having to go through recovery, you must use both the WAIT BEFORE START and WITH CHECKPOINT LOG NO COPY clauses. The WAIT BEFORE START clause ensures that the rollback log is empty, and the WITH CHECKPOINT LOG NO COPY clause ensures that the checkpoint log is empty. If either of these files is missing, then recovery is required. You can use WITH CHECKPOINT LOG RECOVER as an alternative to the WAIT BEFORE START and WITH CHECKPOINT LOG NO COPY clauses if you do not need to recover the database you backed up.

Syntax - Image backup

An image backup creates copies of each of the database files, in the same way as the Backup utility (dbbackup). By default, the Backup utility makes the backup on the client computer, but you can specify the -s option to create the backup on the database server when using the Backup utility. For the BACKUP DATABASE statement, however, the backup can only be made on the database server.

Optionally, only the database file(s) or transaction log can be saved. The transaction log may also be renamed or truncated after the backup has completed.

Alternatively, you can specify an empty string as a directory to rename or truncate the log without copying it first. This is useful in a replication environment where space is a concern. You can use this feature with an event handler on transaction log size to rename the transaction log when it reaches a given size, and with the delete_old_logs option to delete the transaction log when it is no longer needed.

To restore from an image backup, copy the saved files back to their original locations and reapply the transaction logs.

Syntax - Archive backup

An archive backup creates a single file holding all the required backup information.

To restore a database from an archive backup, use the RESTORE DATABASE statement.

If a RESTORE DATABASE statement references an archive file containing only a transaction log, the statement must specify a file name for the location of the restored database file, even if that file does not exist. For example, to restore from an archive that only contains a transaction log to the directory `C:\MYNEWDB`, the RESTORE DATABASE statement is:

```
RESTORE DATABASE 'c:\\temp\\mynewdb\\my.db' FROM archive-root
```

> ⚠ Caution
>
> Backup copies of the database and transaction log must not be changed in any way. If there were no transactions in progress during the backup, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG RECOVER or WITH CHECKPOINT LOG NO COPY, you can check the validity of the backup database using read-only mode or by validating a copy of the backup database.

However, if transactions were in progress, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG COPY, the database server must perform recovery on the database when you start it. Recovery modifies the backup copy, which is not desirable.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

Privileges

You must have the BACKUP DATABASE system privilege.

Side Effects

Causes a checkpoint.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Back up the current database and the transaction log, each to a different file, and rename the existing transaction log. An image backup is created.

```
BACKUP DATABASE
DIRECTORY 'c:\\temp\\backup'
TRANSACTION LOG RENAME;
```

The option to rename the transaction log is useful, especially in replication environments where the old transaction log is still required.

Rename the transaction log without making a copy:

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

Execute the BACKUP DATABASE statement with a dynamically constructed directory name:

```
CREATE EVENT NightlyBackup
```

```
SCHEDULE
START TIME '23:00' EVERY 24 HOURS
HANDLER
BEGIN
    DECLARE dest LONG VARCHAR;
    DECLARE day_name CHAR(20);

    SET day_name = DATENAME( WEEKDAY, CURRENT DATE );
    SET dest = 'd:\\backups\\' || day_name;
    BACKUP DATABASE DIRECTORY dest
        TRANSACTION LOG RENAME;
END;
```

Related Information

Database Backup and Recovery
Types of Backup
Transaction Log File Management in a Database Mirroring System
Backing up a Database Using the BACKUP DATABASE Statement
progress_messages Option
RESTORE DATABASE Statement [page 1339]

1.4.4.37 BEGIN Statement

Specifies a compound statement.

Syntax

```
[ statement-label : ]
BEGIN [ [ NOT ] ATOMIC ]
   [ local-declaration; ... ]
   statement-list
   [ EXCEPTION [ exception-case ... ] ]
END [ statement-label ]
```

```
local-declaration :
variable-declaration
| cursor-declaration
| exception-declaration
| temporary-table-declaration
```

```
exception-case :
WHEN exception-name [, ... ] THEN statement-list
| WHEN OTHERS THEN statement-list
```

```
variable-declaration and exception-declaration : see the DECLARE statement
```

```
cursor-declaration : see the DECLARE CURSOR statement
```

```
temporary-table-declaration : see the DECLARE LOCAL TEMPORARY TABLE statement
```

Parameters

statement-label

If the ending `statement-label` is specified, it must match the beginning `statement-label`. The LEAVE statement can be used to resume execution at the first statement after the compound statement. The compound statement that is the body of a procedure or trigger has an implicit label that is the same as the name of the procedure or trigger.

ATOMIC clause

An atomic statement is a statement that is executed completely or not at all. For example, an UPDATE statement that updates thousands of rows might encounter an error after updating many rows. If the statement does not complete, all changes revert back to their original state. Similarly, if you specify that the BEGIN statement is atomic, the statement is executed either in its entirety or not at all.

local-declaration

Immediately following the BEGIN, a compound statement can have local declarations for objects that only exist within the compound statement. A compound statement can have a local declaration for a variable, a cursor, a temporary table, or an exception. Local declarations can be referenced by any statement in that compound statement, or in any compound statement nested within it. Local declarations of the compound statement are visible to the exception handlers for the statement. Local declarations are not visible to other procedures that are called from within a compound statement.

Remarks

The body of a procedure or trigger is a compound statement. Compound statements can also be used in control statements within a procedure or trigger.

A compound statement allows one or more SQL statements to be grouped together and treated as a unit. A compound statement starts with the keyword BEGIN and ends with the keyword END.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

BEGIN, which identifies a compound statement, comprises part of optional ANSI/ISO SQL Language Feature P002.

Example

The body of a procedure or trigger is a compound statement.

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
   DECLARE err_notfound EXCEPTION FOR
      SQLSTATE '02000';
   DECLARE curThisCust CURSOR FOR
      SELECT CompanyName, CAST(
            sum( SalesOrderItems.Quantity *
            Products.UnitPrice ) AS INTEGER) VALUE
      FROM GROUPO.Customers
            LEFT OUTER JOIN SalesOrders
            LEFT OUTER JOIN SalesOrderItems
            LEFT OUTER JOIN Products
      GROUP BY CompanyName;
   DECLARE ThisValue INT;
   DECLARE ThisCompany CHAR( 35 );
   SET TopValue = 0;
   OPEN curThisCust;
   CustomerLoop:
   LOOP
      FETCH NEXT curThisCust
         INTO ThisCompany, ThisValue;
      IF SQLSTATE = err_notfound THEN
         LEAVE CustomerLoop;
      END IF;
      IF ThisValue > TopValue THEN
         SET TopValue = ThisValue;
         SET TopCompany = ThisCompany;
      END IF;
   END LOOP CustomerLoop;
   CLOSE curThisCust;
END;
```

The example below declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```
BEGIN
   DECLARE v1 INT = 5
   DECLARE v2, v3 CHAR(10) = 'abc'
            // ...
END
```

Related Information

Stored Procedures, Triggers, Batches, and User-defined Functions
Error and Warning Handling
Exception Handling and Atomic Compound Statements

Exception Handlers
Atomic Compound Statements
DECLARE Statement [page 1057]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
DECLARE LOCAL TEMPORARY TABLE Statement [page 1066]
CONTINUE Statement [page 818]
SIGNAL Statement [SP] [page 1405]
RESIGNAL Statement [SP] [page 1338]
RAISERROR Statement [page 1315]
BEGIN Statement [T-SQL] [page 787]

1.4.4.38 BEGIN Statement [T-SQL]

Specifies a compound statement.

Syntax

```
BEGIN
statement-list
END
```

```
statement-list :
sql-statement
| variable-declaration
| cursor-declaration
| temporary-table-declaration
```

```
variable-declaration : see the DECLARE statement
```

```
cursor-declaration : see the DECLARE CURSOR statement
```

```
temporary-table-declaration : see the DECLARE LOCAL TEMPORARY TABLE statement
```

Parameters

statement-list

A list of statements and declarations.

Remarks

A BEGIN statement allows one or more SQL statements to be grouped together and treated as a unit, and starts with the keyword BEGIN and ends with the keyword END.

Error handling is different for Transact-SQL compound statements.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

BEGIN, which identifies a compound statement, comprises part of optional ANSI/ISO SQL Language Feature P002.

Example

The example below declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```
BEGIN
   DECLARE v1 INT = 5
   DECLARE v2, v3 CHAR(10) = 'abc'
   // ...
END
```

Related Information

Error Handling in Transact-SQL Procedures
Stored Procedures, Triggers, Batches, and User-defined Functions
Transact-SQL-compatible Databases
DECLARE Statement [page 1057]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
DECLARE LOCAL TEMPORARY TABLE Statement [page 1066]
BEGIN Statement [page 784]

CONTINUE Statement [page 818]
GOTO Statement [page 1191]
RAISERROR Statement [page 1315]

1.4.4.39 BEGIN PARALLEL WORK Statement

Saves time when executing a list of CREATE INDEX or LOAD TABLE statements on a computer with multiple logical processors.

Syntax

```
BEGIN PARALLEL WORK
statement-list
END PARALLEL WORK
```

```
statement-list:
  list of CREATE INDEX statements
| list of LOAD TABLE statements
```

Parameters

CREATE INDEX statement See the CREATE INDEX statement for information and restrictions about running the CREATE INDEX statement inside the BEGIN PARALLEL WORK statement. The CREATE TEXT INDEX statement is not supported.

LOAD TABLE statement

See the LOAD TABLE statement for information and restrictions about running the LOAD TABLE statement inside the BEGIN PARALLEL WORK statement.

Remarks

The BEGIN PARALLEL WORK statement can improve performance by executing in parallel a list of CREATE INDEX or a list of LOAD TABLE statements.

The number of statements that can execute at the same time is limited by:

- the number of available logical processors on the computer that the database server runs on.
- the settings of the -gtc and -gta database server options and the Processor Affinity option of the sa_server_option system procedure.
- the setting of the max_parallel_statements option.

The BEGIN PARALLEL WORK statement executes atomically. If one statement inside the BEGIN PARALLEL WORK statement fails, then the entire statement rolls back. The BEGIN PARALLEL WORK statement, including the statements listed inside of it, is logged to the transaction log.

Granularity is at the table level.

Privileges

You must have the privileges necessary to execute the statements in the list.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard Not in the standard. Vendor extension.

Example

The following example creates two indexes using the BEGIN PARALLEL WORK statement:

```
BEGIN PARALLEL WORK
    CREATE INDEX L_SHIPDATE_IDX
            ON LINEITEM(l_shipdate);
    CREATE INDEX O_ORDERDATE_IDX
            ON ORDERS(o_orderdate);
END PARALLEL WORK;
```

The following example loads three tables into the database by using the BEGIN PARALLEL WORK statement:

```
BEGIN PARALLEL WORK
    LOAD TABLE dba.Part
        FROM 'D:\\data\\part.tbl'
        FORMAT 'ASCII'
        QUOTES OFF ESCAPES ON STRIP OFF HEXADECIMAL OFF
        DELIMITED BY '|'
        ORDER OFF;
    LOAD TABLE dba.Supplier
        FROM 'D:\\data\\supplier.tbl'
        FORMAT 'ASCII'
        QUOTES OFF ESCAPES ON STRIP OFF HEXADECIMAL OFF
        DELIMITED BY '|'
        ORDER OFF;
    LOAD TABLE dba.Partsupp
        FROM 'D:\\data\\partsupp.tbl'
        FORMAT 'ASCII'
        QUOTES OFF ESCAPES ON STRIP OFF HEXADECIMAL OFF
        DELIMITED BY '|'
        ORDER OFF;
END PARALLEL WORK;
```

Related Information

Improving Performance by Executing a List of CREATE INDEX or a List of LOAD TABLE Statements Concurrently
CREATE INDEX Statement [page 886]
LOAD TABLE Statement [page 1247]
Unload Utility (dbunload)
max_parallel_statements Option

1.4.4.40 BEGIN SNAPSHOT Statement

Starts a snapshot at a specified period in time for use with snapshot isolation transactions.

Syntax

```
BEGIN SNAPSHOT
```

Remarks

By default, when a transaction begins, the database server defers creating the snapshot until the application causes the first row of a table to be fetched. You can use the BEGIN SNAPSHOT statement to start the snapshot earlier within the transaction. The database server creates a snapshot when the BEGIN SNAPSHOT statement is executed by a snapshot transaction.

The statement fails and returns an error when either of the following conditions is met:

- support for snapshots transactions has not been enabled for the database.
- a snapshot has already been started for the current transaction.

This statement is also useful for non-snapshot transactions because it allows them to start a snapshot that can be used later in the transaction for a statement-level snapshot operation.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Snapshot Isolation
allow_snapshot_isolation Option

1.4.4.41 BEGIN TRANSACTION Statement [T-SQL]

Begins a user-defined transaction.

Syntax

```
BEGIN TRAN[SACTION] [ transaction-name ]
```

Remarks

The optional parameter `transaction-name` is the name assigned to this transaction. It must be a valid identifier. Use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

When executed inside a transaction, the BEGIN TRANSACTION statement increases the nesting level of transactions by one. The nesting level is decreased by a COMMIT statement. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

Both Adaptive Server Enterprise and SQL Anywhere have two transaction modes.

The default Adaptive Server Enterprise transaction mode, called unchained mode, commits each statement individually, unless an explicit BEGIN TRANSACTION statement is executed to start a transaction. In contrast, the ANSI/ISO SQL Standard compatible chained mode only commits a transaction when an explicit COMMIT is executed or when a statement that carries out an autocommit (such as a data definition statement) is executed.

You can control the mode by setting the chained database option. The default setting for ODBC and Embedded SQL connections in SQL Anywhere is On, in which case the database server runs in chained mode. (ODBC users should also check the AutoCommit ODBC setting). The default for TDS connections is Off.

In unchained mode, a transaction is implicitly started before any data retrieval or manipulation statement. These statements include: DELETE, INSERT, OPEN, FETCH, SELECT, and UPDATE. You must still explicitly end the transaction with a COMMIT or ROLLBACK statement.

You cannot alter the setting of the chained option within a transaction.

> ⚠ Caution
>
> When calling a stored procedure, you should ensure that it operates correctly under the required transaction mode.

The current nesting level is held in the global variable @@trancount. The @@trancount variable has a value of zero before the first BEGIN TRANSACTION statement is executed, and only a COMMIT executed when @@trancount is equal to one makes changes to the database permanent.

You should not rely on the value of @@trancount for more than keeping track of the number of explicit BEGIN TRANSACTION statements that have been executed.

When Adaptive Server Enterprise starts a transaction implicitly, the @@trancount variable is set to 1. SQL Anywhere does not set the @@trancount value to 1 when a transaction is started implicitly. Instead, the SQL Anywhere @@trancount variable has a value of zero before any BEGIN TRANSACTION statement (even though there is a current transaction), while in Adaptive Server Enterprise (in chained mode) it has a value of 1.

For transactions starting with a BEGIN TRANSACTION statement, @@trancount has a value of 1 in both SQL Anywhere and Adaptive Server Enterprise after the first BEGIN TRANSACTION statement. If a transaction is implicitly started with a different statement, and a BEGIN TRANSACTION statement is then executed, @@trancount has a value of 2 in both SQL Anywhere, and Adaptive Server Enterprise after the BEGIN TRANSACTION statement.

A ROLLBACK statement without a transaction or savepoint name always rolls back statements to the outermost BEGIN TRANSACTION (explicit or implicit) statement, and cancels the entire transaction.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

BEGIN TRANSACTION is supported by Adaptive Server Enterprise.

Example

The following batch reports successive values of @@trancount as 0, 1, 2, 1, and 0. The values are printed in the database server messages window.

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT
PRINT @@trancount
COMMIT
PRINT @@trancount
```

Related Information

Savepoints Within Transactions
COMMIT Statement [page 811]
ROLLBACK Statement [page 1355]
SAVEPOINT Statement [page 1361]
isolation_level Option
chained Option

1.4.4.42 BREAK Statement [T-SQL]

Exits a compound statement or loop.

Syntax

```
BREAK
```

Remarks

The BREAK statement is a control statement that allows you to leave a loop. Execution resumes at the first statement after the loop.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In this example, the BREAK statement breaks the WHILE loop if the most expensive product has a price above $50. Otherwise, the loop continues until the average price is greater than or equal to $30:

```
WHILE ( SELECT AVG( UnitPrice ) FROM Products ) < $30
BEGIN
   UPDATE GROUPO.Products
   SET UnitPrice = UnitPrice + 2
   IF ( SELECT MAX(UnitPrice) FROM Products ) > $50
      BREAK
END
```

Related Information

Stored Procedures, Triggers, Batches, and User-defined Functions
WHILE Statement [T-SQL] [page 1487]
CONTINUE Statement [page 818]
BEGIN Statement [page 784]

1.4.4.43 CALL Statement

Invokes a procedure.

Syntax

Specify argument by position

```
[variable = ] CALL procedure-name ( [ expression, ... ] ) [ AS USER
{ string | variable } IDENTIFIED BY { string | variable } ]
```

Specify argument by keyword format

```
[variable = ] CALL procedure-name ( [ parameter-name = expression, ... ] )
[ AS USER { string | variable } IDENTIFIED BY { string | variable } ]
```

Parameters

AS USER ... IDENTIFIED BY clause This optional clause calls a procedure or function as a different user. The database server verifies that the user ID and password provided are valid, and then executes the procedure or function as the specified user. The invoker of the procedure is the specified user. Upon exiting the procedure or function, the user context is restored to its original state.

> **i** Note
>
> All string values must be enclosed in single quotes; otherwise the database server interprets them as variable names.

Remarks

The CALL statement invokes a procedure that was previously created with a CREATE PROCEDURE statement. When the procedure completes, any INOUT or OUT parameter value is copied back.

> **i** Note
>
> The AS USER ... IDENTIFIED BY clause only applies to the CALL statement and is not supported for procedures in the FROM clause or functions in the select list.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Database-scope variables must not be specified as INOUT or OUT parameters when calling a procedure.

The argument list can be specified by position or by using keyword format. By position, the arguments match up with the corresponding parameter in the parameter list for the procedure (DEFAULT can be used for an optional parameter). By keyword, the arguments are matched up with the named parameters.

Procedure arguments can be assigned default values in the CREATE PROCEDURE statement, and missing parameters are assigned the default value. If no default is set, and an argument is not provided, an error is given.

Inside a procedure, a CALL statement can be used in a DECLARE statement when the procedure returns result sets.

Subqueries and spatial method calls are not allowed as arguments to a stored procedure in a CALL statement.

Procedures can return an integer value (for example, as a status indicator) using the RETURN statement. You can save this return value in a variable using the equality sign as an assignment operator:

If the procedure being called returns an INT and the value is NULL, then the error status value, 0, is returned instead. There is no way to differentiate between this case and the case of an actual value of 0 being returned.

> **i** Note
>
> Use of this statement to invoke a function is deprecated. If you have a function you want to call, consider using an assignment statement to invoke the function and assign its result to a variable. For example:
>
> ```
> DECLARE varname INT;
> SET varname=test( );
> ```

Privileges

The user calling the procedure, or the user specified by the AS USER ... IDENTIFIED BY clause, must be the owner of the procedure, or have one of the following privileges:

- EXECUTE privilege on the procedure
- EXECUTE ANY PROCEDURE system privilege

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Core Feature. The use of the RETURN statement to return a value from a stored procedure is not part of the standard; the ANSI/ISO SQL Standard supports return values only for SQL-invoked functions, not for procedures. Default values for stored procedure arguments are not in the standard.

Example

The following example creates a procedure to return the number of orders placed by the customer whose ID is supplied, creates a variable to hold the result, calls the procedure, and displays the result.

```
CREATE PROCEDURE OrderCount ( IN customer_ID INT, OUT Orders INT )
BEGIN
   SELECT COUNT( GROUPO.SalesOrders.ID )
   INTO Orders
```

```
   FROM GROUPO.Customers
   KEY LEFT OUTER JOIN SalesOrders
   WHERE Customers.ID = customer_ID;
END
go
 -- Create a variable to hold the result
CREATE VARIABLE Orders INT
go
-- Call the procedure, FOR customer 101
CALL OrderCount ( 101, Orders )
go
--  Display the result
SELECT Orders FROM SYS.DUMMY
go
```

The following example uses the AS USER...IDENTIFIED BY clause to execute a procedure as a different user:

1. Create three sample users by executing the following statements:

   ```
   CREATE USER u IDENTIFIED BY pwdforu;
   CREATE USER u1 IDENTIFIED BY pwdforu1;
   CREATE USER u2 IDENTIFIED BY pwdforu2;
   ```

2. Create and populate two tables by executing the following statements:

   ```
   CREATE TABLE u1.t1 (c1 INT);
   CREATE TABLE u2.t2 (c1 INT);
   INSERT INTO u1.t1 VALUES(1);
   INSERT INTO u2.t2 VALUES(2);
   COMMIT;
   ```

3. Create two stored procedures by executing the following statements:

   ```
   CREATE PROCEDURE u1.p1( OUT ret INT, OUT inv_user CHAR(128), OUT exec_user
   CHAR(128) )
   SQL SECURITY INVOKER
   BEGIN
      SELECT c1 INTO ret FROM t1;
      SET inv_user = invoking_user;
      SET exec_user = executing_user;
   END;
   ```

   ```
   CREATE PROCEDURE u2.p2( OUT ret INT, OUT inv_user CHAR(128), OUT exec_user
   CHAR(128) )
   SQL SECURITY DEFINER
   BEGIN
      SELECT c1 INTO ret FROM t2;
      SET inv_user = invoking_user;
      SET exec_user = executing_user;
   END;
   ```

4. Create a third procedure that calls the first procedure as user u1, using string values in the AS USER...IDENTIFIED BY clause:

   ```
   CREATE PROCEDURE u.p1( OUT ret INT, OUT inv_user CHAR(128), OUT exec_user
   CHAR(128) )
   SQL SECURITY DEFINER
   BEGIN
      CALL u1.p1 ( ret, inv_user, exec_user ) AS USER 'u1' IDENTIFIED BY
   'pwdforu1';
   END;
   ```

5. Create a fourth procedure that calls the second procedure as user u, using variable values in the AS USER...IDENTIFIED BY clause:

```
CREATE PROCEDURE u.p2( IN u CHAR(128), IN p CHAR(128), OUT ret INT, OUT
inv_user CHAR(128), OUT exec_user CHAR (128) )
SQL SECURITY DEFINER
BEGIN
   CALL u2.p2( ret, inv_user, exec_user ) AS USER u IDENTIFIED BY p;
END;
```

6. Now, if user u logs in and executes `CALL u1.p1( ret, inv_user, exec_user )` or `CALL u2.p2( ret, inv_user, exec_user )`, then user u has permission denied since user u does not have permission to execute u1.p1 or u2.p2. However, user u is able to execute the following procedure:

   ```
   CALL u1.p1( ret, inv_user, exec_user ) AS USER 'u1' IDENTIFIED BY 'pwdforu1';
   ```

   Both the inv_user and exec_user are returned as u1 even though u1.p1 is a SQL SECURITY INVOKER procedure. This is because the database server executes u1.p1 as user u1, even though the user logged in is user u. Because the procedure executes as though u1 is calling it, it can access the non-fully qualified table t1, since u1.t1 exists.
   Similarly, u is able to execute the following procedure:

   ```
   CALL u2.p2( ret, inv_user, exec_user ) AS USER uvar IDENTIFIED BY pvar;
   ```

   Here, uvar is a variable that contains the value 'u2' and pvar is a variable that contains the value 'pwdforu2'. The procedure executes without error and the inv_user and exec_user both come back as u2 and the procedure can access the non-fully qualified table t2.
7. If user u executes the following two procedures, then both calls succeed since the database server accepts the AS USER...IDENTIFIED BY clause nested within other stored procedures:

   ```
   CALL u.p1( ret, inv_user, exec_user );
   ```

   ```
   CALL u.p2( 'u2', 'pwdforu2', ret, inv_user, exec_user );
   ```

Related Information

Result Sets
Stored Procedures, Triggers, Batches, and User-defined Functions
CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [External Call] [page 866]
CREATE FUNCTION Statement [Web Service] [page 874]
CREATE PROCEDURE Statement [page 913]
CREATE PROCEDURE Statement [External Call] [page 921]
CREATE PROCEDURE Statement [Web Service] [page 931]
EXECUTE Statement [T-SQL] [page 1153]

1.4.4.44 CASE Statement

Selects an execution path based on multiple cases.

Syntax

Specify value expressions

```
CASE value-expression
WHEN [ constant | NULL ] THEN statement-list ...
[ WHEN [ constant | NULL ] THEN statement-list ] ...
[ ELSE statement-list ]
END [ CASE ]
```

Specify search conditions

```
CASE
WHEN [ search-condition | NULL] THEN statement-list ...
[ WHEN [ search-condition | NULL] THEN statement-list ] ...
[ ELSE statement-list ]
END [ CASE ]
```

Remarks

CASE statement using value expressions

The CASE statement is a control statement that allows you to choose a list of SQL statements to execute based on the value of an expression. The `value-expression` is an expression that takes on a single value, which may be a string, a number, a date, or other SQL data type. If a WHEN clause exists for the value of `value-expression`, the `statement-list` in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the `statement-list` in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.

If the `value-expression` can be null, use the ISNULL function to replace the NULL `value-expression` with a different expression.

CASE statement using search conditions

With this form, the statements are executed for the first satisfied `search-condition` in the CASE statement. The ELSE clause is executed if none of the `search-conditions` are met.

If the expression can be NULL, use the following syntax for the first `search-condition`:

```
WHEN search-condition IS NULL THEN statement-list
```

Note

Do not confuse the syntax of the CASE statement with that of the CASE expression.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The CASE statement is part of Language Feature P002 (Computational completeness). The use of END alone, rather than END CASE, is not in the standard.

Transact-SQL

The CASE statement is supported by Adaptive Server Enterprise.

Example

The following procedure using a case statement classifies the products listed in the Products table of the sample database into one of shirt, hat, shorts, or unknown:

```
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
   DECLARE prod_name CHAR(20);
   SELECT Name INTO prod_name FROM GROUPO.Products
   WHERE ID = product_ID;
   CASE prod_name
   WHEN 'Tee Shirt' THEN
      SET type = 'Shirt'
   WHEN 'Sweatshirt' THEN
      SET type = 'Shirt'
   WHEN 'Baseball Cap' THEN
      SET type = 'Hat'
   WHEN 'Visor' THEN
      SET type = 'Hat'
   WHEN 'Shorts' THEN
      SET type = 'Shorts'
   ELSE
      SET type = 'UNKNOWN'
   END CASE;
END;
```

The following example uses search conditions to generate a message about product quantity within the SQL Anywhere sample database:

```
CREATE PROCEDURE StockLevel (IN product_ID INT)
BEGIN
```

```
   DECLARE qty INT;
   SELECT Quantity INTO qty FROM GROUPO.Products
   WHERE ID = product_ID;
   CASE
   WHEN qty < 30 THEN
      MESSAGE 'Order Stock' TO CLIENT;
   WHEN qty > 100 THEN
      MESSAGE 'Overstocked' TO CLIENT;
   ELSE
      MESSAGE 'Sufficient stock on hand' TO CLIENT;
   END CASE;
END;
```

Related Information

Unknown Values: NULL
Stored Procedures, Triggers, Batches, and User-defined Functions
ISNULL Function [Miscellaneous] [page 429]
BEGIN Statement [page 784]
CASE Expressions [page 39]
CASE Statement [T-SQL] [page 802]

1.4.4.45 CASE Statement [T-SQL]

Selects an execution path based on multiple cases.

Syntax

Specify a value expression

```
CASE value-expression
WHEN [ constant | NULL ] THEN statement-list ...
[ WHEN [ constant | NULL ] THEN statement-list ] ...
[ ELSE statement-list ]
END
```

Specify a search condition

```
CASE
WHEN [ search-condition | NULL] THEN statement-list ...
[ WHEN [ search-condition | NULL] THEN statement-list ] ...
[ ELSE statement-list ]
END
```

Remarks

Using a value expression

The CASE statement is a control statement that allows you to choose a list of SQL statements to execute based on the value of an expression. The `value-expression` is an expression that takes on a single value, which may be a string, a number, a date, or other SQL data type. If a WHEN clause exists for the value of `value-expression`, the `statement-list` in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the `statement-list` in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.

If the `value-expression` can be null, use the ISNULL function to replace the NULL `value-expression` with a different expression.

Using a search condition

With this form, the statements are executed for the first satisfied `search-condition` in the CASE statement. The ELSE clause is executed if none of the `search-conditions` are met.

If the expression can be NULL, use the following syntax for the first `search-condition`:

```
WHEN search-condition IS NULL THEN statement-list
```

> **i** Note
>
> Do not confuse the syntax of the CASE statement with that of the CASE expression.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The CASE statement is part of Language Feature P002 (Computational completeness). However, the ANSI/ISO SQL Standard requires END CASE to terminate the CASE statement, rather than END alone.

Transact-SQL

Compatible with Adaptive Server Enterprise.

Example

The following procedure using a case statement classifies the products listed in the Products table of the sample database into one of shirt, hat, shorts, or unknown.

```
CREATE PROCEDURE DBA.ProductType( @product_ID INTEGER,@TYPE CHAR(10) OUTPUT ) AS
BEGIN
  DECLARE @prod_name CHAR(20)
  SELECT Name INTO @prod_name FROM GROUPO.Products
    WHERE ID = @product_ID
  IF @prod_name
   = 'Tee Shirt'
    SET @TYPE = 'Shirt'
  ELSE IF @prod_name
   = 'Sweatshirt'
    SET @TYPE = 'Shirt'
  ELSE IF @prod_name
   = 'Baseball Cap'
    SET @TYPE = 'Hat'
  ELSE IF @prod_name
   = 'Visor'
    SET @TYPE = 'Hat'
  ELSE IF @prod_name
   = 'Shorts'
    SET @TYPE = 'Shorts'
  ELSE
    SET @TYPE = 'UNKNOWN'
END;
```

The following example uses a search condition to generate a message about product quantity within the sample database.

```
CREATE PROCEDURE DBA.StockLevel( @product_ID INTEGER ) AS
BEGIN
  DECLARe @qty INTEGER
  SELECT Quantity INTO @qty FROM GROUPO.Products
    WHERE ID = @product_ID
  IF @qty < 30
    MESSAGE 'Order Stock' TO CLIENT
  ELSE IF @qty > 100
    MESSAGE 'Overstocked' TO CLIENT
  ELSE
    MESSAGE 'Sufficient stock on hand' TO CLIENT
END;
```

Related Information

Unknown Values: NULL
Stored Procedures, Triggers, Batches, and User-defined Functions
ISNULL Function [Miscellaneous] [page 429]
BEGIN Statement [page 784]
CASE Expressions [page 39]

1.4.4.46 CHECKPOINT Statement

Checkpoints the database.

Syntax

```
CHECKPOINT
```

Remarks

The CHECKPOINT statement forces the database server to execute a checkpoint. Checkpoints are also performed automatically by the database server according to an internal algorithm. It is not normally required for applications to issue the CHECKPOINT statement.

Privileges

You must have the CHECKPOINT system privilege to perform a checkpoint on a database running on a network server (dbsrv).

No privileges are required to perform a checkpoint on a database running on a personal database server (dbeng17).

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

The CHECKPOINT statement is supported by Adaptive Server Enterprise.

Related Information

Database Backup and Recovery

Checkpoint Logs
checkpoint_time Option
recovery_time Option

1.4.4.47 CLEAR Statement [Interactive SQL]

Closes any open result sets in Interactive SQL.

Syntax

```
CLEAR
```

Remarks

Closes any open result sets and leaves the contents of the SQL Statements pane unchanged

Privileges

None.

Side Effects

Closes the cursor associated with the data being cleared.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Interactive SQL

1.4.4.48 CLOSE statement [ESQL] [SP]

Closes a cursor.

Syntax

```
CLOSE cursor-name
```

```
cursor-name : identifier | hostvar
```

Remarks

This statement closes the named cursor.

The cursor must have been previously opened.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Core Feature. When used in Embedded SQL, the CLOSE statement is part of optional Language Feature B031 (Basic dynamic SQL).

Transact-SQL

Supported by Adaptive Server Enterprise.

Example

The following examples close cursors in Embedded SQL.

```
EXEC SQL CLOSE employee_cursor;
EXEC SQL CLOSE :cursor_var;
```

The following procedure uses a cursor.

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
   DECLARE err_notfound EXCEPTION
      FOR SQLSTATE '02000';
   DECLARE curThisCust CURSOR FOR
   SELECT CompanyName, CAST(    sum(SalesOrderItems.Quantity *
   Products.UnitPrice) AS INTEGER) VALUE
   FROM GROUPO.Customers
   LEFT OUTER JOIN SalesOrders
   LEFT OUTER JOIN SalesOrderItems
   LEFT OUTER JOIN Products
   GROUP BY CompanyName;
DECLARE ThisValue INT;
   DECLARE ThisCompany CHAR(35);
   SET TopValue = 0;
   OPEN curThisCust;
   CustomerLoop:
   LOOP
      FETCH NEXT curThisCust
      INTO ThisCompany, ThisValue;
         IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop;
         END IF;
         IF ThisValue > TopValue THEN
            SET TopValue = ThisValue;
            SET TopCompany = ThisCompany;
         END IF;
      END LOOP CustomerLoop;
   CLOSE curThisCust;
END
```

Related Information

OPEN Statement [ESQL] [SP] [page 1286]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
PREPARE Statement [ESQL] [page 1308]

1.4.4.49 COMMENT Statement

Stores a comment for a database object in the system tables.

Syntax

```
COMMENT ON {
```

```
    COLUMN [ owner.]table-name.column-name
  | CERTIFICATE certificate-name
  | DBSPACE dbspace-name
  | EVENT [ owner.]event-name
  | EXTERNAL ENVIRONMENT environment-name
  | EXTERNAL [ ENVIRONMENT ] OBJECT object-name
  | FOREIGN KEY [ owner.]table-name.key-name
  | INDEX [ [ owner.] table.]index-name
  | INTEGRATED LOGIN integrated-login-id
  | JAVA CLASS java-class-name
  | JAVA JAR java-jar-name
  | KERBEROS LOGIN "client-Kerberos-principal"
  | LDAP SERVER ldapua-server-name
  | LOGIN POLICY policy-name
  | MATERIALIZED VIEW [ owner.]materialized-view-name
  | MIRROR SERVER mirror-server-name
  | ODATA PRODUCER name
  | PRIMARY KEY ON [ owner.]table-name
  | PROCEDURE [ owner.]procedure-name
  | PUBLICATION [ owner.] publication-name
  | REMOTE MESSAGE TYPE remote-message-type-name
  | ROLE role-name
  | SEQUENCE sequence-name
  | SERVICE web-service-name
  | SPATIAL REFERENCE SYSTEM srs-name
  | SPATIAL UNIT OF MEASURE uom-identifier
  | SYNCHRONIZATION PROFILE synchronization-profile-name
  | TABLE [ owner.]table-name
  | TEXT CONFIGURATION [ owner.]text-config-name
  | TEXT INDEX text-index-name ON [ owner.]table-name
  | TIME ZONE name
  | TRIGGER [ [ owner.]tablename.]trigger-name
  | USER userid
  | VIEW [ owner.]view-name
}
IS comment
```

```
comment : string | NULL
```

```
environment-name :
JAVA
| PERL
| PHP
| CLR
| C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64
```

Remarks

The COMMENT statement allows you to set a remark (comment) for an object in the database. The COMMENT statement updates remarks listed in the ISYSREMARK system table. You can remove a comment by setting it to NULL. For a comment on an index or trigger, the owner of the comment is the owner of the table on which the index or trigger is defined.

You cannot add comments for local temporary tables.

If you use the *Database Documentation Wizard* to document your database, you have the option to include the comments for procedures, functions, triggers, events, and views in the output.

Privileges

If you have the COMMENT ANY OBJECT system privilege, you can comment on any you can create with the CREATE ANY OBJECT system privilege. If you do not have the COMMENT ANY OBJECT system privilege, you must have the equivalent as noted below:

- For database objects, at least one of the following must be true:
  - you own the object
  - you have the ability to create or alter objects of the same type owned by other users (for example, CREATE ANY TABLE, or ALTER ANY OBJECT)
  - you have the ability to manage objects of that type (for example, MANAGE ANY USER)
- For system roles, you must have the administrative privilege over the role.
- For user-defined roles, you must have the MANAGE ROLES system privilege, or have administrative privilege over the role.
- For Kerberos or integrated logins, you must have the MANAGE ANY USER system privilege.
- For Java classes or jars, you must have the MANAGE ANY EXTERNAL OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

Not supported by Adaptive Server Enterprise.

Example

The following examples show how to add and remove a comment.

1. Add a comment to the Employees table:

```
COMMENT ON TABLE GROUPO.Employees
IS 'Employee information';
```

2. Remove the comment from the Employees table:

```
COMMENT
ON TABLE GROUPO.Employees
IS NULL;
```

To view the comment set for an object, use a SELECT statement. The following statement retrieves the comment set for the ViewSalesOrders view in the SQL Anywhere sample database.

```
SELECT remarks
FROM SYSTAB t, SYSREMARK r
WHERE t.object_id = r.object_id
AND t.table_name = 'ViewSalesOrders';
```

Related Information

Documenting a Database (SQL Central)
ALTER ODATA PRODUCER Statement [page 699]
DROP ODATA PRODUCER Statement [page 1108]

1.4.4.50 COMMIT Statement

Makes changes to the database permanent, or terminates a user-defined transaction.

Syntax

Committing work

```
COMMIT [ WORK ]
```

Committing at a transaction level

```
COMMIT TRAN[SACTION] [ transaction-name ]
```

Parameters

transaction-name

An optional name assigned to this transaction. It must be a valid identifier. Use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

The following options control the behavior of the COMMIT statement.

- cooperative_commit_timeout option
- cooperative_commits option
- delayed_commits option

- delayed_commit_timeout option

You can use the Commit connection property to return the number of commits on the current connection.

Remarks

Committing work

The COMMIT statement ends a transaction and makes all changes made during this transaction permanent in the database.

All data definition statements automatically perform a commit. For information, see the Side effects listing for each SQL statement.

The COMMIT statement fails if the database server detects any invalid foreign keys. This behavior makes it impossible to end a transaction with any invalid foreign keys. Usually, foreign key integrity is checked on each data manipulation operation. However, if the database option wait_for_commit is set On or a particular foreign key was defined with a CHECK ON COMMIT clause, the database server delays integrity checking until the COMMIT statement is executed.

The use of COMMIT alone is equivalent to COMMIT WORK.

Committing at a transaction level

You can use BEGIN TRANSACTION and COMMIT TRANSACTION statements in pairs to construct nested transactions. Nested transactions are similar to savepoints. When executed as the outermost of a set of nested transactions, the statement makes changes to the database permanent. When executed inside a transaction, the COMMIT TRANSACTION statement decreases the nesting level of transactions by one. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

Committing at a transaction level is a Transact-SQL extension.

Privileges

None.

Side Effects

Closes all cursors except those opened WITH HOLD.

Deletes all rows of declared temporary tables on this connection, unless they were declared using ON COMMIT PRESERVE ROWS.

If the database is not using a transaction log, each COMMIT operation causes an implicit checkpoint.

Standards

ANSI/ISO SQL Standard

Committing work is a Core Feature. Committing at a transaction level is a Transact-SQL extension.

Example

The following statement commits the current transaction:

```
COMMIT;
```

The following Transact-SQL batch reports successive values of @@trancount as 0, 1, 2, 1, 0.

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
go
```

Related Information

Permanent Data Changes
BEGIN TRANSACTION Statement [T-SQL] [page 792]
wait_for_commit Option
auto_commit Option [Interactive SQL]
commit_on_exit Option [Interactive SQL]
Keyboard Shortcuts (Interactive SQL)
SAVEPOINT Statement [page 1361]
List of Connection Properties
BEGIN TRANSACTION Statement [T-SQL] [page 792]
PREPARE TO COMMIT Statement [page 1310]
ROLLBACK Statement [page 1355]

1.4.4.51 CONFIGURE Statement [Interactive SQL]

Opens the Interactive SQL *Options* window.

Syntax

```
CONFIGURE
```

Remarks

The CONFIGURE statement opens the Interactive SQL *Options* window. This window displays the current settings of all Interactive SQL options. It does not display or allow you to modify database options. You can configure Interactive SQL settings in this window.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Interactive SQL
Customizing Interactive SQL
SET OPTION Statement [page 1387]

1.4.4.52 CONNECT Statement [ESQL] [Interactive SQL]

Establishes a connection to a database.

Syntax

Shared memory connections

```
CONNECT
[ TO database-server-name ]
[ DATABASE database-file ]
[ AS connection-name ]
[ USER ] userid [ IDENTIFIED BY password ]
```

```
database-server-name, database-file, connection-name, userid, password :
{ identifier | string | hostvar }
```

TCP/IP connections

```
CONNECT USING connect-string
```

```
connect-string : { identifier | string | hostvar }
```

Parameters

AS clause

A connection can optionally be named by specifying the AS clause. This allows multiple connections to the same database, or multiple connections to the same or different database servers, all simultaneously. Each connection has its own associated transaction. You may even get locking conflicts between your transactions if, for example, you try to modify the same record in the same database from two different connections.

For TCP/IP connections, a `connect-string` is a list of parameter settings of the form keyword=value, separated by semicolons, and must be enclosed in single quotes.

Remarks

The CONNECT statement establishes a connection to the database identified by `database-file` running on the database server identified by `database-server-name`. This statement is not supported in procedures, triggers, events, or batches.

Shared memory connections are only supported for connections to database servers running on the same computer. To connect to a local database server using TCP/IP or to a database server running on a different computer, use the syntax for TCP/IP connections.

Embedded SQL behavior

In Embedded SQL, if no `database-server-name` is specified, the default local database server is assumed (the first database server started). If no `database-file` is specified, the first database on the given server is assumed.

The WHENEVER statement, SET SQLCA, and some DECLARE statements do not generate code and may appear before the CONNECT statement in the source file. Otherwise, no statements are allowed until a successful CONNECT statement has been executed.

The user ID and password are used for privilege checks on all dynamic SQL statements.

> **i Note**
>
> For SQL Anywhere, only shared memory connections are supported with Embedded SQL. For UltraLite, both shared memory and TCP/IP connections can be used with Embedded SQL.

Interactive SQL behavior

If no database or server is specified in the CONNECT statement, Interactive SQL remains connected to the current database, rather than to the default server and database. If a database name is specified without a server name, Interactive SQL attempts to connect to the specified database on the current server. If a server name is specified without a database name, Interactive SQL connects to the default database on the specified server.

For example, if the following batch is executed while connected to a database, the two tables are created in the same database.

```
CREATE TABLE t1( c1 int );
CONNECT DBA IDENTIFIED BY passwd;
CREATE TABLE t2 (c1 int );
```

No other database statements are allowed until a successful CONNECT statement has been executed.

When Interactive SQL is run in windowed mode, you are prompted for any missing connection parameters.

When Interactive SQL is running in command-prompt mode (-nogui is specified when you start Interactive SQL from a command line) or batch mode, or if you execute CONNECT without an AS clause, an unnamed connection is opened. If there is another unnamed connection already opened, the old one is automatically closed. Otherwise, existing connections are not closed when you execute a CONNECT statement.

Multiple connections are managed through the concept of a current connection. After a successful connect statement, the new connection becomes the current one. To switch to a different connection, use the SET CONNECTION statement. The DISCONNECT statement is used to drop connections.

When connecting to Interactive SQL, specifying CONNECT [USER] `userid` is the same as executing a SETUSER WITH OPTION `userid` statement.

In Interactive SQL, the connection information (including the database name, your user ID, and the database server) appears in the title bar above the SQL Statements pane. If you are not connected to a database, Not Connected appears in the title bar.

> **i Note**
>
> Both syntaxes are valid with Interactive SQL except that Interactive SQL does not support the `hostvar` argument.

This SQL statement is not supported for SAP HANA databases.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Shared memory connections is an optional ANSI/ISO SQL Language Feature F771. TCP/IP connections is not in the standard.

Transact-SQL

Both syntaxes are supported by Adaptive Server Enterprise.

Example

The following are examples of CONNECT usage within Embedded SQL.

```
EXEC SQL CONNECT AS :conn_name
USER :userid IDENTIFIED BY :password;
EXEC SQL CONNECT USER "DBA" IDENTIFIED BY "passwd";
```

The following examples assume that the SQL Anywhere sample database has already been started.

Connect to a database from Interactive SQL. Interactive SQL prompts for a user ID and a password.

```
CONNECT;
```

Connect to the default database as user DBA from Interactive SQL. Interactive SQL prompts for a password.

```
CONNECT USER "DBA";
```

Connect to the sample database as user DBA from Interactive SQL.

```
CONNECT
TO demo17
```

```
USER DBA
IDENTIFIED BY sql;
```

Connect to the sample database using a connection string, from Interactive SQL.

```
CONNECT
USING 'UID=DBA;PWD=sql;DBN=demo';
```

Related Information

Alphabetical List of Connection Parameters
Interactive SQL
GRANT CONNECT Statement [page 1197]
DISCONNECT Statement [ESQL] [Interactive SQL] [page 1085]
SET CONNECTION Statement [Interactive SQL] [ESQL] [page 1378]
SETUSER Statement [page 1403]
Troubleshooting: Connections

1.4.4.53 CONTINUE Statement

Restarts a loop.

Syntax

```
CONTINUE [ statement-label ]
```

Remarks

The CONTINUE statement is a control statement that restarts a loop. Execution continues at the first statement in the loop.

When CONTINUE appears within a set of statements using Transact-SQL, do not use `statement-label`.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

CONTINUE without a statement label is supported by SAP Adaptive Server Enterprise.

Example

The following fragment shows how the CONTINUE statement restarts a loop. This example displays the odd numbers between 1 and 10.

```
BEGIN
   DECLARE i INT;
   SET i = 0;
   lbl:
   WHILE i < 10 LOOP
      SET i = i + 1;
      IF mod( i, 2 ) = 0 THEN
         CONTINUE lbl
      END IF;
      MESSAGE 'The value ' || i || ' is odd.' TO CLIENT;
   END LOOP lbl;
END
```

Related Information

Stored Procedures, Triggers, Batches, and User-defined Functions
LOOP Statement [page 1269]
WHILE Statement [T-SQL] [page 1487]
FOR Statement [page 1166]
BEGIN Statement [page 784]

1.4.4.54 CREATE CERTIFICATE Statement

Adds or replaces a certificate in the database from the given file or string. To create a certificate, use the Certificate Creation utility (createcert).

Syntax

```
CREATE [ OR REPLACE ] CERTIFICATE certificate-name
FROM { certificate-string | variable-name | FILE file-name }
```

Parameters

FROM clause

This clause specifies a file, string, or variable containing a certificate.

Remarks

The CREATE CERTIFICATE statement adds or replaces a certificate in the database from the given file, string, or variable. The file, string, or variable should contain either a binary DER-format certificate or a text PEM-format certificate. DER-format certificates are converted and stored as PEM certificates.

Certificates that are stored in the database can be used by web service procedures and functions that make secure HTTPS connections to a web server. They can also be used to send secure messages using the xp_startsmtp system procedure.

When you add a certificate, it is added to the ISYSCERTIFICATE system table. Use the corresponding system view SYSCERTIFICATE to view the table.

The CREATE CERTIFICATE statement is not used to create an actual certificate. Use the Certificate Creation utility (createcert) to do this.

Privileges

You must have the MANAGE CERTIFICATES system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a certificate called mycert in the database using the contents of the specified certificate file.

```
CREATE CERTIFICATE mycert
FROM FILE 'C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\Certificates\\rsaroot.crt';
```

Related Information

DROP CERTIFICATE Statement [page 1087]
CREATE PROCEDURE Statement [Web Service] [page 931]
CREATE FUNCTION Statement [Web Service] [page 874]
xp_startsmtp System Procedure [page 1881]
SYSCERTIFICATE System View [page 1900]
sa_certificate_info System Procedure [page 1532]
Certificate Creation Utility (createcert)

1.4.4.55 CREATE DATABASE Statement

Creates a database.

Syntax

```
CREATE DATABASE db-filename-string
KEY DERIVATION ITERATIONS number
DBA USER userid-string
DBA PASSWORD password-string
[ create-option ... ]
```

```
create-option :
    [ ACCENT { RESPECT | IGNORE | FRENCH } ]
    [ ASE [ COMPATIBLE ] ]
    [ BLANK PADDING { ON | OFF } ]
    [ CASE { RESPECT | IGNORE } ]
    [ CHECKSUM { ON | OFF } ]
    [ COLLATION collation-label[ ( collation-tailoring-string ) ] ]
    [ DATABASE SIZE size { KB | MB | GB | PAGES | BYTES } ]
```

```
    [ ENCODING encoding-label ]
    [ ENCRYPTED [ TABLE ] { algorithm-key-spec | OFF } ]
    [ JCONNECT { ON | OFF } ]
    [ MINIMUM PASSWORD LENGTH positive-integer ]
    [ PAGE SIZE page-size ]
    [ NCHAR COLLATION nchar-collation-label[ ( collation-tailoring-string ) ] ]
    [ SYSTEM PROCEDURE AS DEFINER { ON | OFF } ]
    [ [ TRANSACTION ] LOG { OFF | ON [ log-filename-string ] [ MIRROR mirror-
filename-string ] } ]
```

```
page-size :
2048 | 4096 | 8192 | 16384 | 32768
```

```
algorithm-key-spec :
ON
| [ ON ] KEY key [ ALGORITHM algorithm ]
| [ ON ] ALGORITHM algorithm KEY key
| [ ON ] ALGORITHM 'SIMPLE'
```

```
algorithm :
'AES' | 'AES_FIPS' | 'AES256' | 'AES256_FIPS' | 'AES256CTR' | 'AES256CTR_FIPS' |
'ARIA256' | 'ARIA256CTR'
```

Parameters

CREATE DATABASE

Each of `db-filename-string`, `log-filename-string`, and `mirror-filename-string` consists of an optional path followed by the name of a file. As literal strings, they must be enclosed in single quotes.

- If you specify a path, any backslash characters (\) must be doubled if they are followed by an n or an x. Escaping them prevents them from being interpreted as new line characters (\n) or as hexadecimal numbers (\x), according to the rules for strings in SQL.
  Here are some examples where this is important.

  ```
  CREATE DATABASE 'c:\\temp\\\x41\x42\x43xyz.db'
  DBA USER 'DBA' DBA PASSWORD 'passwd';
  ```

  The initial \\ sequence represents a backslash. The \x sequences represent the characters A, B, and C, respectively. The file name here is `ABCxyz.db`.

  ```
  CREATE DATABASE 'c:\temp\\nest.db'
  DBA USER 'DBA' DBA PASSWORD 'passwd';
  ```

  To avoid having the \n sequence interpreted as a newline character, the backslash is doubled.
  It is always safer to escape the backslash character. For example:

  ```
  CREATE DATABASE 'c:\\my_db.db'
  DBA USER 'DBA' DBA PASSWORD 'passwd'
  LOG ON 'e:\\logdrive\\my_db.log';
  ```

- If you do not specify a path, or a relative path, the database file is created relative to the working directory of the database server. If you specify no path for a transaction log file, the file is created in the

same directory as the database file. Store the database files and the transaction log on separate disks on the computer.

- If you provide no file extension, a file is created with extension `.db` for databases, `.log` for the transaction log, and `.mlg` for the transaction log mirror.
- The directory path is relative to the database server.

You cannot specify utility_db for `db-filename-string`. This name is reserved for the utility database.

ACCENT clause

This clause is used to specify accent sensitivity for the database. Support for this clause is deprecated. Use the collation tailoring options provided for the COLLATION and NCHAR COLLATION clauses to specify accent sensitivity.

The ACCENT clause applies only when using the UCA (Unicode Collation Algorithm) for the collation specified in the COLLATION or NCHAR COLLATION clause. ACCENT RESPECT causes the UCA string comparison to respect accent differences between letters. For example, e is less than é. ACCENT FRENCH is similar to ACCENT RESPECT, except that accents are compared from right to left, consistent with the rules of the French language. ACCENT IGNORE causes string comparisons to ignore accents. For example, e is equal to é.

If accent sensitivity is not specified when the database is created, the default accent sensitivity for comparisons and sorting is *insensitive*, with one exception; for Japanese databases created with a UCA collation, the default accent sensitivity is *sensitive*.

ASE COMPATIBLE clause

Do not create the SYS.SYSCOLUMNS and SYS.SYSINDEXES views. By default, these views are created for compatibility with system tables available in Watcom SQL (version 4 and earlier of this software). These views conflict with the Adaptive Server Enterprise compatibility views dbo.syscolumns and dbo.sysindexes.

BLANK PADDING clause

The database server compares all strings as if they are varying length and stored using the VARCHAR domain. This includes string comparisons involving fixed length CHAR or NCHAR columns. In addition, the database server never trims or pads values with trailing blanks when the values are stored in the database.

By default, the database server treats blanks as significant characters. For example, the value 'a ' (the character 'a' followed by a blank) is not equivalent to the single-character string 'a'. Inequality comparisons also treat a blank as any other character in the collation.

If blank padding is enabled (specifying BLANK PADDING ON), the semantics of string comparisons more closely follow the ANSI/ISO SQL standard. With blank-padding enabled, the database server ignores trailing blanks in any comparison.

In the example above, an equality comparison of 'a ' to 'a' in a blank-padded database returns TRUE. With a blank-padded database, fixed-length string values are padded with blanks when they are fetched by an application. Whether the application receives a string truncation warning on such an assignment is controlled by the ansi_blanks connection option.

CASE clause

This clause is used to specify case sensitivity for the database. Support for this clause is deprecated. Use the collation tailoring options provided for the COLLATION and NCHAR COLLATION clauses to specify case sensitivity.

CASE RESPECT causes case-sensitive string comparisons for all CHAR and NCHAR data types. Comparisons using UCA consider the case of a letter only if the base letters and accents are all equal. For

all other collations, uppercase and lowercase letters are distinct; for example, a is less than A, which is less than b, and so on. CASE IGNORE causes case-insensitive string comparisons. Uppercase and lowercase letters are considered to be exactly equal.

If case sensitivity is not specified when the database is created, default case sensitivity for comparisons and sorting is *insensitive*, with one exception; for Japanese databases created with a UCA collation, default case sensitivity is *sensitive*.

CASE RESPECT is provided for compatibility with the ISO/ANSI SQL standard. Identifiers in the database are always case insensitive, even in case-sensitive databases.

CHECKSUM clause

Checksums are used to determine whether a database page has been modified on disk. When you create a database with global checksums enabled, a checksum is calculated for each page just before it is written to disk. The next time the page is read from disk, the page's checksum is recalculated and compared to the checksum stored on the page. If the checksums are different, then the page has been modified on disk and an error occurs. Databases created with global checksums enabled can also be validated using checksums. You can check whether a database was created with global checksums enabled by executing the following statement:

```
SELECT DB_PROPERTY ( 'Checksum' );
```

This query returns ON if global checksums are turned on, otherwise, it returns OFF. Global checksums are turned on by default, so if the CHECKSUM clause is omitted, ON is applied.

Regardless of the setting of this clause, the database server always enables write checksums for databases running on storage devices such as removable drives, to help provide early detection if the database file becomes corrupt. The database server also calculates checksums for critical pages during validation activities.

For databases that do not have global checksums enabled, you can enable write checksums by using the -wc options.

COLLATION clause

The collation specified by the COLLATION clause is used for sorting and comparison of character data types (CHAR, VARCHAR, and LONG VARCHAR). The collation provides character comparison and ordering information for the encoding (character set) being used. If the COLLATION clause is not specified, the database server chooses a collation based on the operating system language and encoding.

The collation can be chosen from the list of collations that use the SQL Anywhere Collation Algorithm (SACA), or it can be the Unicode Collation Algorithm (UCA). If UCA is specified, also specify the ENCODING clause.

It is important to choose your collation carefully. It cannot be changed after the database has been created.

Optionally, you can specify collation tailoring options (`collation-tailoring-string`) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in parentheses, following the collation name. For example, `... CHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'`.

DATABASE SIZE clause

Use this optional clause to set the initial size of the database file. You can use KB, MB, GB, or PAGES to specify units of kilobytes, megabytes, gigabytes, or pages respectively.

Specifying the file size at creation time is a way of preallocating space for the file. This helps reduce the risk of running out of space on the drive the database is located on. As well, it can help improve performance by increasing the amount of data that can be stored in the database before the database server needs to grow the database, which can be a time-consuming operation.

DBA USER and DBA PASSWORD clauses

Use these clauses to specify a DBA user ID and password for the database.

By default, passwords must be a minimum length of 6 characters unless the MINIMUM PASSWORD LENGTH clause is specified and set to a different value. Passwords should be composed of 7-bit ASCII characters. Other characters may not work correctly if the server cannot convert from the client character set to UTF-8.

ENCODING clause

Most collations specified in the COLLATION clause dictate both the encoding (character set) and ordering. For those collations, the ENCODING clause should not be specified. However, if the value specified in the COLLATION clause is UCA (Unicode Collation Algorithm), use the ENCODING clause to specify a locale-specific encoding and get the benefits of the UCA for comparison and ordering. The ENCODING clause may specify UTF-8 or any single-byte encoding for CHAR data types. ENCODING may not specify a multibyte encoding other than UTF-8.

If you choose the UCA collation, you can optionally specify collation tailoring options.

If COLLATION is set to UCA and ENCODING is not specified, then the database server uses UTF-8.

ENCRYPTED or ENCRYPTED TABLE clause

Encryption makes stored data undecipherable. Use the ENCRYPTED keyword (without TABLE) when you want to encrypt the entire database. Use the ENCRYPTED TABLE clause when you only want to enable table encryption. Enabling table encryption means that the tables that are subsequently created or altered using the ENCRYPTED clause are encrypted using the settings you specified at database creation.

There are two levels of database and table encoding: simple obfuscation and strong encryption. Obfuscation is not encryption, and someone with cryptographic expertise could decipher the data. Strong encryption ensures that the data is unreadable and virtually undecipherable.

For simple obfuscation, specify ENCRYPTED ON ALGORITHM SIMPLE, or ENCRYPTED ALGORITHM SIMPLE, or specify the ENCRYPTED ON clause without specifying an algorithm or key.

For strong encryption, specify ENCRYPTED ON ALGORITHM with a 128-bit or 256-bit AES or ARIA algorithm, and the KEY clause to specify an encryption key. Choose a value for your key that is at least 16 characters long, contains a mix of uppercase and lowercase, and includes numbers, letters, and special characters. A key can be specified as either a string or a variable name.

> ⚠ Caution
>
> For strongly encrypted databases, be sure to store a copy of the key in a safe location. If you lose the encryption key there is no way to access the data, even with the assistance of Technical Support. The database must be discarded and you must create a new database.

You can also create an encrypted copy of an existing database by using the CREATE ENCRYPTED DATABASE statement.

JCONNECT clause

To allow the jConnect JDBC driver access to system catalog information, specify JCONNECT ON. This clause installs the system objects that provide jConnect support. Specify JCONNECT OFF to exclude the

jConnect system objects. You can still use JDBC, as long as you do not access system information. JCONNECT is ON by default.

KEY DERIVATION ITERATIONS clause

Sets the number of number of KDF (key derivation function) iterations to use when encrypting the database using AES (or AES256) encryption. Specify a number between 1000 and 100,000. The default is 2000. A large number of iterations provides better security, but a very large number of iterations could cause the database to take longer to start, particularly on slower machines. However, once the database is running, there is no performance difference compared to other encrypted databases.

Once this setting is set, it cannot be altered. However, you can use the CREATE ENCRYPTED DATABASE to create a copy of the database and specify a different number of iterations.

MINIMUM PASSWORD LENGTH clause

Use this clause to set the minimum password length. If this clause is not specified, then the default minimum password length for a new database is 6.

PAGE SIZE clause

The page size for a database can be 2048, 4096, 8192, 16384, or 32768 bytes. The default page size is 4096 bytes. The 2048 page size is deprecated. Large databases generally obtain performance benefits from a larger page size, but there can be additional overhead associated with the large page sizes.

For example:

```
CREATE DATABASE 'c:\\temp\\my_db.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
PAGE SIZE 4096;
```

> **i Note**
>
> The page size cannot be larger than the page size used by the current server. The server page size is taken from the first set of databases started, or is set on the server command line using the -gp option.

NCHAR COLLATION clause

The collation specified by the NCHAR COLLATION clause is used for sorting and comparing national character data types (NCHAR, NVARCHAR, and LONG NVARCHAR). The collation provides character ordering information for the UTF-8 encoding (character set) used for national characters. If the NCHAR COLLATION clause is not specified, the database server uses the Unicode Collation Algorithm (UCA). The only other allowed collation is UTF8BIN, which provides a binary ordering of all characters whose encoding is greater than 0x7E.

Optionally, you can specify collation tailoring options (`collation-tailoring-string`) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in a quoted string following the collation name. For example, `... NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'`. If you specify the ACCENT or CASE clause and a collation tailoring string that contains settings for case and accent, the values of the ACCENT and CASE clauses are used as defaults only.

> **i Note**
>
> When you specify the UCA collation, all collation tailoring options are supported. For all other collations, only the case sensitivity tailoring option is supported.
>
> Databases created with collation tailoring options cannot be started using a pre-10.0.1 database server.

SYSTEM PROCEDURE AS DEFINER { ON | OFF } clause

The SYSTEM PROCEDURE AS DEFINER clause specifies whether to execute pre-16.0 system procedures that perform privileged tasks with the privileges of the invoker or the definer (owner). ON means that these system procedures are executed with the privileges of the definer (owner). OFF means these system procedures are executed with the privileges of the invoker.

If this clause is not specified, the default is to run these procedures with the privileges of the invoker.

This setting does not impact user-defined procedures, or any system procedures introduced in version 16.0 or later.

[TRANSACTION] LOG clause

The transaction log is a file where the database server logs all changes made to the database. The transaction log plays a key role in backup and recovery, and in data replication. The default is LOG ON.

The MIRROR option of the LOG clause allows you to provide a file name if you are using a transaction log mirror. A transaction log mirror is an identical copy of a transaction log, usually maintained on a separate device, for greater protection of your data. By default, the database server does not use a transaction log mirror.

Remarks

Creates a database file with the supplied name and attributes. The database is stored as an operating system file. This statement is not supported in procedures, triggers, events, or batches.

You must be connected to a database to create another database. For example, connect to the utility database.

The account under which the database server is running must have write permissions on the directories where files are created.

Messages sent to the client indicate what type of database encryption is used for the database. If encryption is used, the algorithm being used is also displayed.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side Effects

An operating system file is created.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

The CREATE DATABASE statement is supported by Adaptive Server Enterprise, though with different clauses.

Example

The following statement creates a database file named `temp.db` in the `C:\temp` directory:

```
CREATE DATABASE 'c:\\temp\\temp.db'
DBA USER 'DBA' DBA PASSWORD 'passwd';
```

The following statement creates a database file named `mydb.db` in the `C:\temp` directory.

```
CREATE DATABASE 'C:\\temp\\mydb.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
TRANSACTION LOG ON
CASE IGNORE
PAGE SIZE 4096
ENCRYPTED OFF
BLANK PADDING OFF;
```

The following statement creates a database using code page 1252 and uses the UCA for both CHAR and NCHAR data types. Accents and case are respected during comparison and sorting.

```
CREATE DATABASE 'c:\\temp\\uca.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
COLLATION 'UCA'
ENCODING 'CP1252'
NCHAR COLLATION 'UCA'
ACCENT RESPECT
CASE RESPECT;
```

The following statement creates a database, `myencrypteddb.db`, that is encrypted using simple obfuscation:

```
CREATE DATABASE 'c:\\temp\\myencrypteddb.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
ENCRYPTED ON;
```

The following statement creates a database, `mystrongencryptdb.db`, that is encrypted using the key gh67AB2 (strong encryption):

```
CREATE DATABASE 'c:\\temp\\mystrongencryptdb.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
ENCRYPTED ON KEY 'gh67AB2';
```

The following statement creates a database, `myobfuscatedtabledb.db`, with table encryption enabled using simple obfuscation. Notice the keyword TABLE inserted after ENCRYPTED to indicate table encryption instead of database encryption:

```
CREATE DATABASE 'c:\\temp\\myobfuscatedtabledb.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
ENCRYPTED TABLE ON;
```

The following statement creates a database, `myobfuscatedtabledb.db`, with table encryption enabled using simple encryption:

```
CREATE DATABASE 'c:\\temp\\myobfuscatedtabledb.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
ENCRYPTED TABLE ON ALGORITHM 'SIMPLE';
```

The following statement creates a database file named `mydb.db` that uses collation 1252LATIN1. The NCHAR collation is set to UCA, with the locale set to es, and has case sensitivity and accent sensitivity enabled:

```
CREATE DATABASE 'c:\\temp\\my2.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
COLLATION '1252LATIN1(case=respect)'
NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)';
```

The following statement creates a database with a Greek collation:

```
CREATE DATABASE 'c:\\temp\\mydb.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
COLLATION '1253ELL';
```

The following statement creates a database named `mydb.db` with a transaction log mirror:

```
CREATE DATABASE 'c:\\mydb.db'
DBA USER 'DBA' DBA PASSWORD 'passwd'
TRANSACTION LOG ON 'mydb.log'
MIRROR 'd:\\mydb.mlg';
```

Related Information

Running Pre-16.0 System Procedures as Invoker or Definer
Escape Sequences [page 13]
Corruption Detection Using Checksums
Table Encryption
The Utility Database (utility_db)
Collation Considerations
Simple Obfuscation Versus Strong Encryption
International Languages and Character Sets
ALTER DATABASE Statement [page 662]
CREATE ENCRYPTED DATABASE Statement [page 840]
Collation Tailoring Options
Recommended Character Sets and Collations
Encryption Algorithm Aliases

1.4.4.56 CREATE DBSPACE Statement

Defines a new database space and creates the associated database file.

Syntax

```
CREATE DBSPACE dbspace-name AS filename
```

Parameters

dbspace-name

Specify a name for the dbspace. This is not the actual database file name, which you specify using `filename`. `dbspace-name` is an internal name you can refer to, for example in statements and procedures. You cannot use the following names for a dbspace because they are reserved for predefined dbspaces: system, temporary, temp, translog, and translogmirror.

An error is returned if you specify a value that contains a period (.).

filename

Specify a name for the database file, including, optionally, the path to the file. If no path is specified, the database file is created in the same location (directory) as the main database file. If you specify a different location, the path is relative to the database server. The backslash (\ ) is an escape character in SQL strings, so each backslash must be doubled.

The `filename` parameter must be either a string literal or a variable.

Remarks

The CREATE DBSPACE statement creates a new database file. When a database is created, it is composed of one file. All tables and indexes created are placed in that file. CREATE DBSPACE adds a new file to the database. This file can be on a different disk drive than the main file, which means that the database can be larger than one physical device.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

For each database, there is a limit of twelve dbspaces in addition to the main file.

Each object, such as a table or index, is contained entirely within one dbspace. The IN clause of the CREATE statement specifies the dbspace into which an object is placed. Objects are put into the system database file by default. You can also specify which dbspace tables are created in by setting the default_dbspace option before you create the tables.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side Effects

Automatic commit. Automatic checkpoint.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a dbspace called libbooks in the `c:\` directory. A subsequent CREATE TABLE statement creates a table called LibraryBooks in the libbooks dbspace.

```
CREATE DBSPACE libbooks
AS 'c:\\library.db';
CREATE TABLE LibraryBooks (
  title char(100),
  author char(50),
  isbn char(30),
) IN libbooks;
```

Related Information

Strings [page 11]
Predefined Dbspaces
Additional Dbspaces Considerations
DROP DBSPACE Statement [page 1092]
default_dbspace Option

1.4.4.57 CREATE DECRYPTED DATABASE Statement

Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.

Syntax

```
CREATE DECRYPTED DATABASE newfile
FROM oldfile
[ KEY key ]
```

Parameters

FROM clause

Use this clause to specify the name of the database to copy (`oldfile`).

KEY clause

Use this clause to specify the encryption key needed to decrypt the database. You can specify either a string or a variable name for the key. You do not specify the KEY clause if the existing database was encoded with simple obfuscation, which does not require a key.

Remarks

The CREATE DECRYPTED DATABASE statement produces a new database file (`newfile`), and does not replace or remove the original database file (`oldfile`).

All encrypted tables in `oldfile` are not encrypted in `newfile`, and table encryption is not enabled.

Note

For databases created with SQL Anywhere 12 or later, the ISYSCOLSTAT, ISYSUSER, and ISYSEXTERNLOGIN system tables always remain encrypted to protect the data from unauthorized access.

If `oldfile` uses a transaction log or transaction log mirror, the files are renamed `newfile.log` and `newfile.mlg`, respectively.

If `oldfile` contains dbspace files, a D (decrypted) is added to the file name. For example, when you execute the CREATE DECRYPTED DATABASE statement, the file `mydbspace.dbs` is changed to `mydbspace.dbsD`.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located.

You cannot execute this statement on a database that requires recovery. This statement is not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are decrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The first statement below creates an AES256-encrypted copy of the *demo.db* called `demoEncrypted.db`. The second statement creates a decrypted copy of `demoEncrypted.db` called `demoDecrypted.db`.

```
CREATE ENCRYPTED DATABASE 'demoEncrypted.db'
   FROM 'demo.db'
   KEY 'Sd8f6654*Mnn'
   ALGORITHM 'AES256';
CREATE DECRYPTED DATABASE 'demoDecrypted.db'
   FROM 'demoEncrypted.db'
   KEY 'Sd8f6654*Mnn';
```

Related Information

Database Encryption and Decryption
CREATE ENCRYPTED DATABASE Statement [page 840]
CREATE ENCRYPTED FILE Statement [page 843]
CREATE DECRYPTED FILE Statement [page 834]

1.4.4.58 CREATE DECRYPTED FILE Statement

Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

Syntax

```
CREATE DECRYPTED FILE newfile
FROM oldfile KEY key
```

Parameters

FROM clause

Lists the file name of the encrypted file.

KEY clause

Lists the key required to access the encrypted file. The key can be either a string or a variable name.

Remarks

The CREATE DECRYPTED DATABASE statement is the recommended method for decrypting a database. If the CREATE DECRYPTED DATABASE statement fails, then use the CREATE DECRYPTED FILE statement. The CREATE DECRYPTED FILE statement is often used when you have to decrypt a database for technical support purposes. You can also use this statement to decrypt an associated database file, such as a transaction log, transaction log mirror, or dbspace files.

In addition to decrypting the database file, the CREATE DECRYPTED DATABASE statement automatically decrypts any associated files, such as the transaction log, transaction log mirror, and dbspace files. If you use the CREATE DECRYPTED FILE statement, you must decrypt the associated files individually.

The original database file must be strongly encrypted using an encryption key. The resulting file is an exact copy of the encrypted file, without encryption and therefore requiring no encryption key.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

If a database is decrypted using this statement, then the corresponding transaction log file (and any dbspaces) must also be decrypted to use the database.

If a database requiring recovery is decrypted, its transaction log file must also be decrypted and recovery on the new database is necessary. The name of the transaction log file remains the same in this process, so if the database and transaction log file are renamed, then you need to run dblog -t on the resulting database.

You cannot use this statement on a database that has table encryption enabled. If you have tables you want to decrypt, use the NOT ENCRYPTED clause of the ALTER TABLE statements to decrypt them.

> **i Note**
>
> For databases created with SQL Anywhere 12 or later, the ISYSCOLSTAT, ISYSUSER, and ISYSEXTERNLOGIN system tables always remain encrypted to protect the data from unauthorized access to the database file.

This statement is not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are decrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example decrypts a fictitious encrypted database called encContacts, and creates a new unencrypted database called contacts.

```
CREATE DECRYPTED FILE 'contacts.db'
FROM 'encContacts.db'
KEY 'Sd8f6654*Mnn';
```

Related Information

ALTER TABLE Statement [page 742]
CREATE ENCRYPTED FILE Statement [page 843]

CREATE DECRYPTED DATABASE Statement [page 832]
CREATE ENCRYPTED DATABASE Statement [page 840]
-gu Database Server Option

1.4.4.59 CREATE DOMAIN Statement

Creates a domain in a database.

Syntax

```
CREATE { DOMAIN | DATATYPE } domain-name [ AS ] data-type
[ [ NOT ] NULL ]
[ DEFAULT default-value ]
[ CHECK ( condition ) ]
[ AS USER user-name ]
```

```
domain-name : identifier
```

```
data-type :  built-in data type, with precision and scale, or another domain
```

Parameters

DOMAIN | DATATYPE clause

It is recommended that you use CREATE DOMAIN, rather than CREATE DATATYPE, because CREATE DOMAIN is defined in the ANSI/ISO SQL Standard.

data-type

Set the data type to one of the builtin data types or to the data type of another domain by specifying that domain name. For example:

```
CREATE DOMAIN a INT;
CREATE DOMAIN b a;
```

You can also specify a %TYPE or %ROWTYPE attribute to set the data type to the data type of a column or row in a table or view. However, specifying a table reference variable for the %ROWTYPE (TABLE REF (`table-reference-variable`) %ROWTYPE) is not allowed.

NULL clause

This clause allows you to specify the nullability of a domain. When a domain is used to define a column, nullability is determined as follows:

- Nullability specified in the column definition.
- Nullability specified in the domain definition.
- If the nullability was not explicitly specified in either the column definition or the domain definition, then the setting of the allow_nulls_by_default option is used.

CHECK clause

When creating a domain with a CHECK constraint, you can use a variable name prefixed with the @ sign in the CHECK constraint's search condition. When the data type is used in the definition of a column, such a variable is replaced by the column name. This allows a domain's CHECK constraint to be applied to each table column defined with that domain.

AS USER clause

Specifies the owner of the object.

Remarks

Domains are aliases for built-in data types, including precision and scale values where applicable. They improve convenience and encourage consistency in the database.

Domains are objects within the database. Their names must conform to the rules for identifiers. Domain names are always case insensitive, as are built-in data type names, but they are not collation-insensitive. For example, in the Turkish collation, the domain name "image" can be used (because it was created using lowercase letters) but not the domain "IMAGE". The letter case that is guaranteed to work is the letter case in which the type was created and this can be seen by looking at the type_name column of the SYS.SYSUSERTYPE table.

The user who creates a data type is automatically made the owner of that data type. No owner can be specified in the CREATE DATATYPE statement. The domain name must be unique, and all users can access the data type without using the owner as prefix.

Domains can have CHECK conditions and DEFAULT values, and you can indicate whether the data type permits NULL values or not. These conditions and values are inherited by any column defined on the domain. Any conditions or values explicitly specified in the column definition override those specified for the domain.

The AS USER clause is generated into database unload scripts to annotate the creator of the domain since the creator is recorded in the SYSUSERTYPE system view. Otherwise, it is of no importance.

Privileges

You must have the CREATE DATATYPE or CREATE ANY OBJECT system privilege to create domains owned by you. You cannot create domains owned by others.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Domain support is optional ANSI/ISO SQL Language Feature F251.

Example

Some columns in a database table could be used for people's names and others to store addresses. You might define the following domains.

```
CREATE DOMAIN person CHAR(30) NOT NULL;
CREATE DOMAIN address CHAR(35);
```

The address domain differs from the person domain in the number of characters it can contain. Also, a column of type person cannot contain a NULL value.

Integer values are commonly used as unique identifiers for rows in tables. The following statement creates a domain named identifier, which is an unsigned integer that does not allow NULL values, and which is set to automatically increment by default.

```
CREATE DOMAIN identifier UNSIGNED INT
NOT NULL
DEFAULT AUTOINCREMENT;
```

Having defined these domains, you can use them much as you would the built-in data types. You can use these definitions to define a table, as follows. You need the CREATE TABLE privilege to execute the following statement.

```
CREATE TABLE myCustomers (
   ID        identifier  PRIMARY KEY,
   Name      person,
   Street    address
);
```

In the above example, the table's primary key ID is an automatically incrementing non-NULL unsigned integer value (of course, primary keys can never be NULL).

Many of your tables may require similar identifier columns. Instead of specifying the same set of attributes each time, it is much more convenient to create the identifier domain and use this everywhere. The same can be said for the names of persons and their addresses. Maintenance of table schema is simplified by the use of domains.

Domains can be defined in terms of other domains. The following is an example..

```
CREATE DOMAIN simple_identifier UNSIGNED INT;
CREATE DOMAIN identifier simple_identifier NOT NULL DEFAULT AUTOINCREMENT;
```

The CREATE DOMAIN statements in the following example create domains based on the data types of the Name and Street columns of the myCustomers table. These new domains are used to define columns in the myCustomers2 table.

```
CREATE DOMAIN customers_name myCustomers.Name%TYPE NOT NULL;
CREATE DOMAIN customers_street myCustomers.Street%TYPE;

CREATE TABLE myCustomers2 (
   ID        identifier  PRIMARY KEY,
   Name      customers_name,
   Street    customers_street
```

```
);
```

In this example, only the data types are selected from myCustomers.Name and myCustomers.Street so the NOT NULL attribute is specified for the customers_name domain.

The order of creation of domains and tables is important. Obviously, the customers_name and customers_street domains cannot be created before the myCustomers table. Also, any adjustment to the definition of columns in the myCustomers table after the domains have been defined is not automatically carried over to the domain definitions.

When you create a domain, you can provide a CHECK constraint to ensure that no inappropriate values are entered into any column of this type. The following statement creates a domain named phone_number, which uses a regular expression within a CHECK constraint to ensure that the string has a properly formatted North American phone number of 12 characters, consisting of a 3-digit area code that does not start with 0 or 1, a 3-digit exchange that does not start with 0 or 1, and a 4-digit number separated by either dashes or blanks.

```
CREATE DOMAIN phone_number CHAR(12) NULL
CHECK( @phone_number REGEXP '([2-9][0-9]{2}-[2-9][0-9]{2}-[0-9]{4})|([2-9][0-9]
{2}\s[2-9][0-9]{2}\s[0-9]{4})');
```

The following statement creates a ROW domain named MyRow, which can hold a row of data:

```
CREATE DOMAIN MyRow ROW( a INT, b INT );
```

The new MyRow domain can then be referenced in SQL statements. For example:

```
CREATE FUNCTION Swap( @parm MyRow )
   RETURNS (MyRow)
     BEGIN
        DECLARE @ret MyRow;
        SET @ret = ROW( @parm.b, @parm.a );
        RETURN @ret;
      END;
```

The following statement creates an ARRAY domain named MyArray, which can hold an array of rows:

```
CREATE DOMAIN MyArray ARRAY( 10 ) OF ROW( a INT, b INT );
```

Given the earlier declaration for MyRow, this example could just as well have been written as follows:

```
CREATE DOMAIN MyArray ARRAY( 10 ) OF MyRow;
```

Related Information

%TYPE and %ROWTYPE Attributes [page 115]
allow_nulls_by_default Option
DROP DOMAIN Statement [page 1093]
SQL Data Types [page 129]

1.4.4.60 CREATE ENCRYPTED DATABASE Statement

Creates an obfuscated or encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table obfuscation or encryption enabled.

Syntax

Create an encrypted copy of a database

```
CREATE ENCRYPTED DATABASE newfile
FROM oldfile
[ KEY newkey ]
[ ALGORITHM algorithm ]
[ KEY DERIVATION ITERATIONS number ]
[ OLD KEY oldkey ]
```

```
algorithm :
    'SIMPLE'
    | 'AES'
    | 'AES256'
    | 'AES_FIPS'
    | 'AES256_FIPS'
```

Create a copy of a database with table encryption enabled

```
CREATE ENCRYPTED TABLE DATABASE newfile
FROM oldfile
[ KEY newkey ]
[ ALGORITHM algorithm ]
[ KEY DERIVATION ITERATIONS number ]
[ OLD KEY oldkey ]
```

Parameters

CREATE ENCRYPTED DATABASE clause

Specifies the name for the new database.

CREATE ENCRYPTED TABLE DATABASE clause

Specifies the name for the new database. The new database is not encrypted, but has table encryption enabled.

FROM clause

Specifies the name of the original database file (`oldfile`).

KEY clause

Specifies the encryption key for `newfile`. The key can be either a string or a variable name. Not required for ALGORITHM 'SIMPLE'.

ALGORITHM clause

Specifies the encoding algorithm to use for `newfile`. You may choose between SIMPLE obfuscation or some form of AES encryption. If you specify a KEY clause but do not specify the ALGORITHM clause, AES

(128-bit encryption) is used by default. If you specify 'SIMPLE' for `algorithm`, you do not specify a KEY clause.

KEY DERIVATION ITERATIONS

Specifies the number of times that the encryption key is hashed. Specify a whole number between 1 and 1000. The default value is 2, which is 2000 iterations. The higher the number, the better security. If you are running the database on a slow computer, then a very high number of iterations could result in the database taking longer to start (once the database is running, there is no performance impact).

OLD KEY clause

Use this clause to specify the encryption key for `oldfile`. The key can be either a string or a variable name. This clause is only required if `oldfile` is encrypted using some form of AES encryption.

Remarks

Use this statement to create an obfuscated or encrypted copy of an existing database, including all transaction logs and dbspaces. Note that obfuscated databases are not strongly encrypted and provide no security against skilled and determined attempts to gain access to the data.

You can also use this statement to create a copy of a database and enable table encryption in the copy.

The database file `oldfile` can be an unencrypted database, an encrypted database, or a database with table encryption enabled.

Creating an encrypted copy of a database takes an existing database, `oldfile`, and creates an encrypted copy of it, `newfile`.

Creating a copy of a database with table encryption enabled takes an existing database, `oldfile`, and creates a copy of it, `newfile`, with table encryption enabled. When you use this syntax, any tables encrypted in `oldfile` are encrypted in `newfile` as well. If no tables were encrypted in `oldfile`, but you want to encrypt them, then execute an ALTER TABLE...ENCRYPTED statement on each table you want to encrypt.

Neither syntax replaces or removes `oldfile`.

If `oldfile` uses transaction log or transaction log mirror files, they are renamed `newfile.log` and `newfile.mlg` respectively.

If `oldfile` contains dbspace files, an E (for encrypted) is added to the file name. For example, when you execute the CREATE ENCRYPTED DATABASE statement, the file `mydbspace.dbs` is changed to `mydbspace.dbsE`.

You can use this statement to change the encryption algorithm and key for a database. However, the CREATE ENCRYPTED DATABASE statement produces a new file (`newfile`), and does not replace or remove the previous version of the file (`oldfile`).

When encrypting the database or enabling table encryption in the database, you must specify an encryption key. The software uses Password-Based Key Derivation Function #2 (PBKDF2), which is part of the PKCS#5 standard to protect the key from brute-force attacks. The software repeatedly applies a cryptographic hash to the encryption key. Use the KEY DERIVATION ITERATIONS clause to specify the number of times to apply the hash.

CREATE ENCRYPTED DATABASE and CREATE ENCRYPTED TABLE DATABASE cannot be executed against a database that requires recovery. You must use the CREATE ENCRYPTED FILE statement instead.

These statements are not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are encrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

You can also encrypt an existing database or change an existing encryption key by unloading and reloading the database using the dbunload -an option with either -ek or -ep.

You can also create an encrypted database, or a database with table encryption enabled, using the CREATE DATABASE statement.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates an encrypted copy of the sample database called `demoEnc.db`. The new database is encrypted with AES256 encryption.

```
CREATE ENCRYPTED DATABASE 'demoEnc.db'
   FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere
         17\\Samples\\sample.db'
   KEY 'Sd8f6654*Mnn'
   ALGORITHM 'AES256';
```

The following example creates a copy of the sample database called `demoTableEnc.db`. Table encryption is enabled on the new database. Since a key was specified with no algorithm, AES encryption is used.

```
CREATE ENCRYPTED TABLE DATABASE 'demoTableEnc.db'
```

```
    FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\sample.db'
    KEY 'Sd8f6654';
```

Related Information

Tips on Rebuilding Databases Using the Unload Utility
Database Encryption and Decryption
Table Encryption
Simple Obfuscation Versus Strong Encryption
CREATE DECRYPTED DATABASE Statement [page 832]
CREATE ENCRYPTED FILE Statement [page 843]
CREATE DECRYPTED FILE Statement [page 834]
CREATE DATABASE Statement [page 821]
ALTER TABLE Statement [page 742]
Initialization Utility (dbinit)
-gu Database Server Option

1.4.4.61 CREATE ENCRYPTED FILE Statement

Creates a strongly encrypted copy of a database file when you cannot use the CREATE ENCRYPTED DATABASE statement. Also creates encrypted copies of the transaction log, transaction log mirror, and dbspace files.

Syntax

```
CREATE ENCRYPTED FILE newfile
FROM oldfile
KEY newkey
[ ALGORITHM algorithm ]
[ KEY DERIVATION ITERATIONS number ]
[ OLD KEY oldkey ]
```

```
algorithm :
    'AES'
  | 'AES256'
  | 'AES_FIPS'
  | 'AES256_FIPS'
```

Parameters

FROM clause

Specifies the name of the original database file (`oldfile`).

KEY clause

Specifies the encryption key to use for `newfile`. The key can be either a string or a variable name. This key must be specified.

ALGORITHM clause

Specifies the algorithm used to encrypt `newfile`. If you do not specify an algorithm, AES (128-bit encryption) is used by default.

KEY DERIVATION ITERATIONS

Specifies the number of times that the encryption key is hashed. Specify a whole number between 1 and 1000. The default value is 2, which is 2000 iterations. The higher the number, the better security. If you are running the database on a slow machine, then a very high number of iterations could result in the database taking longer to start (once the database is running, there is no performance impact).

OLD KEY clause

Specifies the encryption key for `oldfile`, if it is encrypted. The key can be either a string or a variable name.

Remarks

The CREATE ENCRYPTED FILE statement is provided for the situation when you need to encrypt a database for technical support purposes and you cannot use CREATE ENCRYPTED DATABASE statement The CREATE ENCRYPTED DATABASE statement is the recommended statement for encrypting a database. However, if the CREATE ENCRYPTED DATABASE statement fails, then you can use the CREATE ENCRYPTED FILE statement. You can also use the CREATE ENCRYPTED FILE statement to create encrypted copies of a transaction log, transaction log mirror, and dbspace files. If you execute the CREATE ENCRYPTED FILE statement against an encrypted database, then you create an encrypted copy of the database with a different encryption key and algorithm.

When encrypting a database, you must execute the CREATE ENCRYPTED FILE statement against the database file as well as against each of the database-related files independently (transaction log, transaction log mirror, dbspace files, if any).

The CREATE ENCRYPTED FILE statement produces a new file (`newfile`), and does not replace or remove the previous version of the file (`oldfile`).

If the database has table encryption enabled, you cannot use the CREATE ENCRYPTED FILE statement; use the CREATE ENCRYPTED DATABASE statement instead.

The CREATE ENCRYPTED FILE statement is not supported in procedures, triggers, events, or batches.

When encrypting the database or enabling table encryption in the database, specify an encryption key. When encrypting the database-related files, specify the same algorithm, key, and iteration count for all files related to the database. The software uses Password-Based Key Derivation Function #2 (PBKDF2), which is part of the PKCS#5 standard to protect the key from brute-force attacks. The software repeatedly applies a cryptographic hash to the encryption key. Use the KEY DERIVATION ITERATIONS clause to specify the number of times to apply the hash.

You can change the number of iterations that are applied to the encryption key by using the -kdi option. The minimum number of iterations is 1000 and the maximum is 1,000,000. The more iterations that you apply the longer it takes to create the encryption key and the longer it takes for a brute-force attack to test a candidate password.

If `oldfile` has dbspaces or transaction log files associated with it and you encrypt those too, you must ensure that the new name and location of those files is stored with the new database. To do so:

- Run dblog -t on the new database to change the name and location of the transaction log.
- Run dblog -m on the new database to change the name and location of the transaction log mirror.
- Execute an ALTER DBSPACE statement on the new database to change the location and name of the dbspace files.

To execute the CREATE ENCRYPTED FILE statement, you must connect to a different database from the one that you are encrypting. For example, connect to the utility database. The database that you that you are encrypting must not be running.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example encrypts the sample database *demo.db* and creates a new database called `demo2.db` that is encrypted with AES_FIPS encryption. The new database file is placed in the server's current working directory.

```
CREATE ENCRYPTED FILE 'demo2.db'
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\demo.db'
   KEY 'Sd8f6654*Mnn'
   ALGORITHM 'AES_FIPS';
```

The following example encrypts the sample database *demo.db* and its transaction log file `demo.log`.

```
CREATE ENCRYPTED FILE 'demo3.db'
   FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere
         17\\Samples\\demo.db'
   KEY 'Sd8f6654*Mnn';
CREATE ENCRYPTED FILE 'demo3.log'
   FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere
         17\\Samples\\demo.log'
   KEY 'Sd8f6654*Mnn';
```

The new database file and transaction log are placed in the server's current working directory. At a command prompt, use the Transaction Log utility (dblog) to set the new transaction log name since the new database file `demo3.db` still references the old transaction log file.

```
dblog -ek Sd8f6654*Mnn -t demo3.log demo3.db
```

To change the encryption key for a database, create a copy of the database file and transaction log using the new key, as shown in the following example:

```
CREATE ENCRYPTED FILE 'c:\\temp\\demo.db'
   FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere
         17\\Samples\\demo.db'
   KEY     'Sd251072*Mnn'
   OLD KEY 'Sd8f6654*Mnn';
CREATE ENCRYPTED FILE 'C:\\temp\\demo.log'
   FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere
         17\\Samples\\demo.log'
   KEY     'Sd251072*Mnn'
   OLD KEY 'Sd8f6654*Mnn';
```

The new database file and transaction log are placed in the specified directory. Now you can archive the old database file and its transaction log, and then move the new database file and transaction log to the same directory where the old files were located.

Related Information

Database Encryption and Decryption
CREATE ENCRYPTED DATABASE Statement [page 840]
CREATE ENCRYPTED DATABASE Statement [page 840]
CREATE DECRYPTED FILE Statement [page 834]
CREATE DECRYPTED DATABASE Statement [page 832]
Unload Utility (dbunload)
Transaction Log Utility (dblog)
-gu Database Server Option

1.4.4.62 CREATE EVENT Statement

Defines an event and its associated handler for automating predefined actions, and to define scheduled actions.

Syntax

```
CREATE [ OR REPLACE ] EVENT [user-name.]event-name
[ TYPE event-type
          [ WHERE trigger-condition [ AND trigger-condition ] ... ]
    | SCHEDULE schedule-spec, ... ]
[ ENABLE | DISABLE ]
[ AT { CONSOLIDATED | REMOTE | ALL } ]
[ FOR { PRIMARY | ALL } ]
[ HANDLER
      BEGIN
...
     END ]
```

```
event-type :
  BackupEnd
| Connect
| ConnectFailed
| DatabaseStart
| DBDiskSpace
| Deadlock
| "Disconnect"
| GlobalAutoincrement
| GrowDB
| GrowLog
| GrowTemp
| LogDiskSpace
| MirrorFailover
| MirrorServerDisconnect
| RAISERROR
| ServerIdle
| TempDiskSpace
```

```
trigger-condition :
event_condition( condition-name ) {
=
| <
| >
| !=
| <=
| >=
} value
```

```
schedule-spec :
[ schedule-name ]
   { START TIME start-time | BETWEEN start-time AND end-time }
   [ EVERY period { HOURS | MINUTES | SECONDS } ]
   [ ON { ( day-of-week, ... ) | ( day-of-month, ... ) } ]
   [ START DATE start-date ]
```

```
event-name : identifier
```

```
schedule-name : identifier
```

```
day-of-week : string
day-of-month : integer
value : integer
period : integer
start-time : time
end-time : time
start-date : date
```

Parameters

event-name

The event name is an identifier. An event has a creator, which is the user creating the event, and the event handler executes with the privileges of that creator. This is the same as stored procedure execution. You cannot create events owned by other users.

user-name

Optionally, specify the name of a user in the system; when the event runs, it runs with the privileges of `user-name`. If this parameter is not specified, the event runs with the privileges of the user who created the event. `user-name` should not be confused with an owner of the event, however; events do not have owners.

OR REPLACE clause

Specifying OR REPLACE (CREATE OR REPLACE EVENT) creates an event or replaces an event with the same name. If the event already exists, then all comments are preserved when you use the OR REPLACE clause, but all existing attributes of the event are dropped.

TYPE clause

You can specify the TYPE clause with an optional WHERE clause, or specify the SCHEDULE.

The `event-type` is one of the listed set of system-defined event types. The event types are case insensitive. To specify the conditions under which this `event-type` triggers the event, use the WHERE clause.

DiskSpace event types

If the database contains an event handler for one of the DiskSpace types, the database server checks the available space on each device associated with the relevant file every 30 seconds.

In the event the database has more than one dbspace, on separate drives, DBDiskSpace checks each drive and acts depending on the lowest available space.

The LogDiskSpace event type checks the location of the transaction log and any transaction log mirror, and reports based on the least available space.

The TempDiskSpace event type checks the amount of temporary disk space.

If the appropriate event handlers have been defined (DBDiskSpace, LogDiskSpace, or TempDiskSpace), the database server checks the available space on each device associated with a database file every 30 seconds. Similarly, if an event has been defined to handle the system event type ServerIdle, the database server notifies the handler when no requests have been processed during the previous 30 seconds.

You can specify the -fc option when starting the database server to implement a callback function when the database server encounters a file system full condition.

GlobalAutoincrement event type

The event fires on *each* insert when the number of remaining values for a GLOBAL AUTOINCREMENT is less than 1% of the end of its range. A typical action for the handler could be to request a new value for the global_database_id option, based on the table and number of remaining values which are supplied as parameters to this event.

You can use the event_condition function with RemainingValues as an argument for this event type.

ServerIdle event type

If the database contains an event handler for the ServerIdle type, the database server checks for server activity every 30 seconds.

Database mirroring event types

The MirrorServerDisconnect event fires when a connection from the primary database server to the mirror server or arbiter server is lost, and the MirrorFailover event fires whenever a server takes ownership of the database.

WHERE clause

The trigger condition determines the condition under which an event is fired. For example, to take an action when the disk containing the transaction log becomes more than 80% full, use the following triggering condition:

```
...
WHERE event_condition( 'LogFreePercent' ) < 20
...
```

The argument to the event_condition function must be valid for the event type.

You can use multiple AND conditions to make up the WHERE clause, but you cannot use OR conditions or other conditions.

You can specify a variable name for the event_condition value.

SCHEDULE clause

This clause specifies when scheduled actions are to take place. The sequence of times acts as a set of triggering conditions for the associated actions defined in the event handler.

You can create more than one schedule for a given event and its associated handler. This permits complex schedules to be implemented. You must provide a `schedule-name` when there is more than one schedule; the `schedule-name` is optional if you provide only a single schedule.

A scheduled event is recurring if its definition includes EVERY or ON; if neither of these reserved words is used, the event executes at most once. An attempt to create a non-recurring scheduled event for which the start time has passed generates an error. When a non-recurring scheduled event has passed, its schedule is deleted, but the event handler is not deleted.

Scheduled event times are calculated when the schedules are created, and again when the event handler completes execution. The next event time is computed by inspecting the schedule or schedules for the event, and finding the next schedule time that is in the future. If an event handler is instructed to run every hour between 9:00 and 5:00, and it takes 65 minutes to execute, it runs at 9:00, 11:00, 1:00, 3:00, and 5:00. If you want execution to overlap, you must create more than one event.

The subclauses of a schedule definition are as follows:

START TIME clause

The first scheduled time for each day on which the event is scheduled. The `start-time` parameter is a string, and cannot be an expression such as `NOW()`. If a START DATE is specified, the START TIME refers to that date and each subsequent day (if the schedule includes EVERY or ON). If no START DATE is specified, the START TIME is on the current day (unless the time has passed) and each subsequent day (if the schedule includes EVERY or ON). The clause START TIME `start-time` is equivalent to BETWEEN `start-time` AND '23:59:59'.

You can specify a variable name for `start-time`.

BETWEEN...AND clause

A range of times during the day outside which no scheduled times occur. The `start-time` and `end-time` parameters are strings, and cannot be expressions such as `NOW()`. If a START DATE is specified, the scheduled times do not occur until that date.

You can specify a variable name for `start-time` and `end-time`.

EVERY clause

An interval between successive scheduled events. Scheduled events occur only after the START TIME for the day, or in the range specified by BETWEEN...AND.

You can specify a variable name for `period`.

ON clause

A list of days on which the scheduled events occur. The default is every day if EVERY is specified. Days can be specified as days of the week or days of the month.

Days of the week are Mon, Tues, and so on. You may also use the full forms of the day, such as Monday. You must use the full forms of the day names if the language you are using is not English, is not the language requested by the client in the connection string, and is not the language which appears in the database server messages window.

Days of the month are integers from 0 to 31. A value of 0 represents the last day of any month.

START DATE clause

The date on which scheduled events are to start occurring. This value is a string, and cannot be an expression such as `TODAY()`. The default is the current date.

You can specify a variable name for `start-date`.

Each time a scheduled event handler is completed, the following actions are taken to calculate the next scheduled time and date for the event:

1. If the EVERY clause is used, find whether the next scheduled time falls on the current day, and is before the end time specified by the BETWEEN...AND clause, if it was specified. If so, that is the next scheduled time.

2. If the next scheduled time does not fall on the current day, find the next date on which the event is to be executed and use the START TIME for that date, or the beginning of the BETWEEN...AND range.

ENABLE | DISABLE clause

By default, event handlers are enabled. When DISABLE is specified, the event handler does not execute even when the scheduled time or triggering condition occurs. A TRIGGER EVENT statement does *not* cause a disabled event handler to be executed.

AT clause

This clause should be used only in the following circumstance: in a SQL Remote setup, use the AT clause against your remote or consolidated databases to restrict the databases at which the event is handled.

If you do not use the AT clause when creating events for SQL Remote, all databases execute the event. When executed on a consolidated database, this statement does not affect remote databases that have already been extracted.

FOR clause

This clause should only be used in the following circumstance: in a database mirroring or read-only scale-out system, use the FOR clause to restrict the databases at which the event is handled.

If you do not use the FOR clause when creating an event for a database in a mirroring or read-only scale-out system, then only the database that is running on the primary server executes the event. The following subclauses are supported:

FOR PRIMARY

The event executes only on the server currently acting as the primary server. The default is the PRIMARY sub clause.

When the FOR PRIMARY clause (or the FOR clause is not specified) is used with the DatabaseStart event type, the event executes when a server becomes the primary server for the database.

FOR ALL

The event executes on all servers in the system.

When the FOR ALL clause is used with the DatabaseStart event type, the event is executed when any database starts. If in a mirroring system the event did not run when the database started (for example, the database was running before the event was created), then the event can execute during a fail over. For example you start a database mirroring system, you create a DatabaseStart event with the FOR ALL clause, and then you stop the primary server, which causes a fail over. In this example, the event executes on the new primary server. The DatabaseStart event will not execute during subsequent fail overs.

HANDLER clause

Each event has one handler.

Remarks

Events can be used for:

Scheduling actions

The database server executes actions on a timed schedule. You can use this capability to complete scheduled tasks such as backups, validity checks, and queries used to add data to reporting tables.

Event handling actions

The database server executes actions when a predefined event occurs. You can use this capability to complete scheduled tasks such as restrict disk space when a disk fills beyond a specified percentage. Event handler actions are committed if errors are not detected during execution, and rolled back if errors are detected.

An event definition includes two distinct pieces. The trigger condition can be an occurrence, such as a disk filling up beyond a defined threshold. A schedule is a set of times, each of which acts as a trigger condition. When a trigger condition is satisfied, the event handler executes. The event handler includes one or more actions specified inside a compound statement (BEGIN... END).

If no trigger condition or schedule specification is supplied, only an explicit TRIGGER EVENT statement can trigger the event. During development, you may want to test event handlers using TRIGGER EVENT, and add the schedule or WHERE clause once testing is complete.

Event errors are logged to the database server message log.

After each execution of an event handler, a COMMIT occurs if no errors occurred. A ROLLBACK occurs if there was an error.

When event handlers are triggered, the database server makes context information, such as the connection ID that caused the event to be triggered, available to the event handler using the event_parameter function.

Event handlers execute on a separate connection, but the separate connection does not count towards the ten-connection limit of the personal database server.

An owner can be specified but is ignored; events do not have owners. Because events do not have owners, no two events can have the same name.

> **i** Note
>
> For parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

If you are creating a new event, then you must have the MANAGE ANY EVENT or CREATE ANY OBJECT system privilege.

If you are replacing an existing event, then you must have on of the following:

- MANAGE ANY EVENT system privilege
- ALTER ANY OBJECT system privilege
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Instruct the database server to perform an incremental backup daily at 1:00 A.M.

```
CREATE EVENT IncrementalBackup
SCHEDULE
 START TIME '1:00 AM' EVERY 24 HOURS
HANDLER
BEGIN
 BACKUP DATABASE DIRECTORY 'c:\\backup'
 TRANSACTION LOG ONLY
 TRANSACTION LOG RENAME MATCH
END;
```

Instruct the database server to perform an automatic backup of the transaction log only, every hour, Monday to Friday between 8 A.M. and 6 P.M.

```
CREATE EVENT HourlyLogBackup
SCHEDULE hourly_log_backup
BETWEEN '8:00AM' AND '6:00PM'
EVERY 1 HOURS ON
    ('Monday','Tuesday','Wednesday','Thursday','Friday')
HANDLER
   BEGIN
      BACKUP DATABASE DIRECTORY 'c:\\database\\backup'
      TRANSACTION LOG ONLY
      TRANSACTION LOG RENAME
   END;
```

Determine when an event is next scheduled to run:

```
SELECT DB_EXTENDED_PROPERTY( 'NextScheduleTime', 'HourlyLogBackup');
```

The following example creates an event that uses a variable for one of the event_condition values, and then creates an event that uses the variable @i1 for the first event_condition value:

```
CREATE VARIABLE @i1 INTEGER;
SET @i1 = 10000;
```

```
CREATE EVENT LogNotifier
TYPE RAISERROR
WHERE event_condition ( 'ErrorNumber' ) <> @i1 AND event_condition
( 'ErrorNumber' ) <> 7
```

```
   HANDLER
      BEGIN
         MESSAGE 'LogNotifier message'
      END;
```

Related Information

System Events
Trigger Conditions for Events
Database Server Logging
BEGIN Statement [page 784]
ALTER EVENT Statement [page 675]
TRIGGER EVENT Statement [page 1441]
EVENT_PARAMETER Function [System] [page 372]
EVENT_CONDITION Function [System] [page 369]
Troubleshooting: Using Database Mirroring System Events to Send Notification Email When Failover Occurs

1.4.4.63 CREATE EXISTING TABLE Statement

Creates a new proxy table, which represents an existing object on a remote server.

Syntax

```
CREATE { EXISTING | VIRTUAL } TABLE [owner.]table-name
[ column-definition, ... ]
AT location-string [ ESCAPE CHARACTER character ]
```

```
column-definition :
column-name data-type NOT NULL
```

```
location-string :
  remote-server-name.[db-name].[owner].object-name
| remote-server-name;[db-name];[owner];object-name
```

Parameters

CREATE { EXISTING | VIRTUAL } TABLE clause

CREATE EXISTING TABLE and CREATE VIRTUAL TABLE are semantically equivalent. CREATE VIRTUAL TABLE is provided for compatibility with SAP HANA.

AT clause

The AT clause specifies the location of the remote object. The AT clause supports the semicolon (;) as a delimiter. If a semicolon is present anywhere in the `location-string` string, then the semicolon is the

field delimiter. If no semicolon is present, then a period is the field delimiter. This behavior allows file names and extensions to be used in the database and owner fields.

When you create a proxy table by using either the CREATE TABLE or the CREATE EXISTING statement, the AT clause includes a location string that is comprised of the following parts:

- The name of the remote server
- The remote catalog
- The remote owner or schema
- The remote table name

The location string can also contain Variable names within the location string are encapsulated within braces.

The string in the AT clause can contain local or global variable names enclosed in braces (for example, {`variable-name`}). These variables are expanded when the database server evaluates `location-string`. The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains `'access;{@myfile};;a1'` indicates that `@myfile` is a SQL variable and that the current contents of the `@myfile` variable should be substituted when the proxy table is created.

When you create a proxy table for an SAP HANA database, the owner, schema, and table name are case sensitive. Use the case as it is specified in the SAP HANA remote database.

ESCAPE CHARACTER clause

The ESCAPE CHARACTER clause allows you to escape a character in a remote server name, catalog name, owner name, schema name, or table name. For example, if the object name contains a character such as period, semicolon, and a brace, it must be escaped by specifying the ESCAPE CHARACTER clause.

`character` can be any single byte character.

Remarks

The CREATE EXISTING TABLE statement creates a new, local, proxy table that maps to a table at an external location. The CREATE EXISTING TABLE statement is a variant of the CREATE TABLE statement. The EXISTING keyword is used with CREATE TABLE to specify that a table already exists remotely and to import its metadata. This syntax establishes the remote table as a visible entity to users. The software verifies that the table exists at the external location before it creates the table.

If the object does not exist (either as a host data file or remote server object), the statement is rejected with an error message.

Index information from the host data file or remote server table is extracted and used to create rows for the ISYSIDX system table. This information defines indexes and keys in server terms and enables the query optimizer to consider any indexes that may exist on this table.

Referential constraints are passed to the remote location when appropriate.

If `column-definitions` are not specified, then the database server derives the column list from the metadata it obtains from the remote table. If `column-definitions` are specified, then the database server

verifies the `column-definitions`. Column names, data types, lengths, the identity property, and null properties are checked for the following conditions:

- Column names must match identically (although case is ignored).
- Data types in the CREATE EXISTING TABLE statement must match or be convertible to the data types of the column on the remote location. For example, a local column data type is defined as money, while the remote column data type is numeric.
- Each column's NULL property is checked. If the local column's NULL property is not identical to the remote column's NULL property, then a warning message is issued, but the statement is not aborted.
- Each column's length is checked. If the length of CHAR, VARCHAR, BINARY, VARBINARY, DECIMAL and/or NUMERIC columns do not match, then a warning message is issued, but the command is not aborted. You may choose to include only a subset of the actual remote column list in your CREATE EXISTING statement.

Privileges

You must have the CREATE PROXY TABLE system privilege to create proxy tables owned by you. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create proxy tables owned by others.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

Supported by Adaptive Server Enterprise. The format of `location-string` is implementation-defined.

Example

Create a proxy table named blurbs for the blurbs table at the remote server server_a.

```
CREATE EXISTING TABLE blurbs
( author_id ID not null,
copy text not null)
AT 'server_a.db1.joe.blurbs';
```

Create a proxy table named blurbs for the blurbs table at the remote server server_a. The database server derives the column list from the metadata it obtains from the remote table.

```
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs';
```

Create a proxy table named rda_employees for the Employees table at the remote server rda.

```
CREATE EXISTING TABLE rda_employees
AT 'rda...Employees';
```

Create a proxy table named rda_employees for a table that is specified by the SQL variable table_name at the remote server rda.

```
CREATE EXISTING TABLE rda_employees
AT 'rda...{table_name}';
```

To utilize the ESCAPE CHARACTER clause, consider the following example:

1. Create two SQL Anywhere databases named test1.db and test2.db.
2. Start both databases on the same server:

   ```
   dbsrv17 -n escape_test test1.db test2.db
   ```

3. Connect to test2 and create the following table:

   ```
   CREATE TABLE "table.with;fun{characters}"(c int);
   INSERT INTO "table.with;fun{characters}" VALUES(100);
   COMMIT;
   ```

4. Disconnect and connect to test1.
5. Create a remote server to test2 as follows:

   ```
   CREATE SERVER test2_server CLASS 'saodbc' USING 'driver=SQL Anywhere
   Native;eng=escape_test;dbn=test2';
   CREATE EXTERNLOGIN localuser TO test2_server REMOTE LOGIN remoteuser
   IDENTIFIED BY remotepwd;
   ```

   > **i Note**
   >
   > localuser is the user ID that is used to log in to test1 while remoteuser and remotepwd are the remote user ID and password needed to log in to test2.

6. The ESCAPE CHARACTER clause can be used to create a proxy table for the remote "table.with;fun{characters}" as follows:

   ```
   CREATE EXISTING TABLE remtab AT 'test2_server;;;table.with!;fun!
   {characters!}'ESCAPE CHARACTER'!';
   ```

   OR

   ```
   CREATE EXISTING TABLE remtab AT 'test2_server...table!.with!;fun!
   {characters!}'ESCAPE CHARACTER'!';
   ```

   Optionally, you can execute a query on the proxy table to ensure you get the expected result set back:

   ```
   SELECT c FROM remtab;
   ```

Related Information

Proxy Table Locations
CREATE TABLE Statement [page 1002]

1.4.4.64 CREATE EXTERNLOGIN Statement

Assigns an alternate login name and password to be used when communicating with a remote server.

Syntax

Create external login for a remote server

```
CREATE EXTERNLOGIN login-name
TO remote-server
[ REMOTE LOGIN remote-user [ IDENTIFIED BY remote-password ] ]
```

Create external login for a remote server (include variables in syntax)

```
CREATE EXTERNLOGIN USER string | variable
SERVER string | variable
[ REMOTE USER string | variable [ IDENTIFIED BY string | variable ] ]
```

Create external login for a directory access server

```
CREATE EXTERNLOGIN login-name
TO remote-server
```

Create external login for a directory access server (include variables in syntax)

```
CREATE EXTERNLOGIN USER string | variable
SERVER string | variable
```

Parameters

login-name Specifies the local user login name. When using integrated logins, `login-name` is the database user to which the Windows user or group is mapped.
TO clause The TO clause specifies the name of the remote server.
REMOTE LOGIN clause The REMOTE LOGIN clause specifies the user account on remote-server for the local user login-name. Values for the REMOTE LOGIN clause are restricted to 128 bytes.
user-name

Specifies the database user name. For remote servers, when using integrated logins, the `user-name` is the database user to which the Windows user or group is mapped. This value can be a string or a variable.
SERVER clause

Specify the name of the remote server or the directory access server.
REMOTE USER clause (remote server)

The REMOTE USER clause specifies the user account on the remote server for the database user name. Values for the REMOTE USER clause are restricted to 128 bytes.

IDENTIFIED BY clause

Specify the remote password for the remote user. The remote user and remote password combination must be valid on the remote server. The This clause applies only to remote servers, not to directory access servers.

If you omit the IDENTIFIED BY clause, then the password is sent to the remote server as NULL. However, if you specify IDENTIFIED BY "" (an empty string), then the password sent is the empty string.

Remarks: Remote servers

CREATE EXTERNLOGIN assigns an alternate login name and password to be used when communicating with a remote server.

Connections to a remote server are first attempted using the current executing user's external login. If this user does not have an external login, then the connection is attempted using the DEFAULT LOGIN credentials. If the remote server was created without a DEFAULT LOGIN, and no external login has been defined for the user, then the connection is attempted with the current executing user's ID and password.

The REMOTE LOGIN clause is required only when the remote server requires a user ID and password for the connection. Having an external login without a remote login allows the DBA to control who can access the remote server and tells the remote access layer that logging in to the remote server does not require a user ID and password.

The password is stored internally in encrypted form. The `remote-server` must be known to the local server by an entry in the ISYSSERVER table.

Sites with automatic password expiration should plan for periodic updates of passwords for external logins.

CREATE EXTERNLOGIN cannot be used from within a transaction.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

> **i** Note
>
> For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Remarks (directory access servers)

By default, database users must have external logins to access the directory access server. However, you can configure the directory access server to remove this requirement by creating a default external login that all users can use.

CREATE EXTERNLOGIN assigns an external login to be used when accessing a directory access server.

CREATE EXTERNLOGIN cannot be used from within a transaction.

Privileges

You must have the MANAGE ANY USER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This fictitious example maps a local user, DBA, to user sa with password Plankton when connecting to the server server1.

```
CREATE EXTERNLOGIN DBA
TO server1
REMOTE LOGIN sa
IDENTIFIED BY Plankton;
```

Related Information

Creating External Logins (SQL Central)
DROP EXTERNLOGIN Statement [page 1095]
CREATE SERVER Statement [page 962]

1.4.4.65 CREATE FUNCTION Statement

Creates a user-defined SQL function in the database.

Syntax

```
CREATE [ OR REPLACE | TEMPORARY ] FUNCTION [ owner.]function-name
( [ parameter, ... ] )
RETURNS data-type
[ SQL SECURITY { INVOKER | DEFINER } ]
[ ON EXCEPTION RESUME ]
[ [ NOT ] DETERMINISTIC ]
compound-statement | AS tsql-compound-statement | AT location-string
```

```
parameter :
   [ IN ] parameter-name data-type [ DEFAULT expression ]
```

```
tsql-compound-statement :
sql-statement
sql-statement
 ...
```

Parameters

OR REPLACE clause

Specifying CREATE OR REPLACE FUNCTION creates a new function, or replaces an existing function with the same name. When a function is replaced, the definition of the function is changed but the existing privileges are preserved.

You cannot use the OR REPLACE clause with temporary functions.

TEMPORARY keyword

Specifying CREATE TEMPORARY FUNCTION means that the function is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Temporary functions can also be explicitly dropped. You cannot perform ALTER, GRANT, or REVOKE on them, and, unlike other functions, temporary functions are not recorded in the catalog or transaction log.

Temporary functions execute with the privileges of their creator (current user) or specified owner. You can specify an owner for a temporary function when:

- the temporary function is created within a permanent stored procedure
- the owner of the temporary function and permanent stored procedure is the same

To drop the owner of a temporary function, you must drop the temporary function first.

Temporary functions can be created and dropped when connected to a read-only database.

You cannot use the OR REPLACE clause with temporary functions.

parameter-name

Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

data-type

The data type of the parameter. Set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

RETURNS clause

Use the RETURNS clause to specify the data type for the result of the function. The RETURNS clause must be the first clause of the statement.

SQL SECURITY clause

The SQL SECURITY clause defines whether the function is executed as the INVOKER (the user who is calling the function), or as the DEFINER (the user who owns the function). The default is DEFINER.

ON EXCEPTION RESUME clause

Use Transact-SQL-like error handling.

[NOT] DETERMINISTIC clause

Use this clause to indicate whether functions are deterministic or non-deterministic. If this clause is omitted, then the deterministic behavior of the function is unspecified (the default).

If a function is declared as DETERMINISTIC, it should return the same value every time it is invoked with the same set of parameters.

If a function is declared as NOT DETERMINISTIC, then it is not guaranteed to return the same value for the same set of parameters. A function declared as NOT DETERMINISTIC is re-evaluated each time it is called in a query. This clause must be used when it is known that the function result for a given set of parameters can vary.

Also, functions that have side effects such as modifying the underlying data should be declared as NOT DETERMINISTIC. For example, here is an example of a function that generates primary key values and is used in an INSERT...SELECT statement; it is declared NOT DETERMINISTIC. The tables that are referenced are fictitious.

```
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
  DECLARE keyval INTEGER;
  UPDATE counter SET x = x + increment;
  SELECT counter.x INTO keyval FROM counter;
  RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;
```

Functions can be declared as DETERMINISTIC if they always return the same value for given input parameters.

compound-statement

A set of SQL statements bracketed by BEGIN and END, and separated by semicolons.

AS clause

`tsql-compound-statement` is a batch of Transact-SQL statements.

AT clause

Create a proxy function on the current database for a remote function specified by `location-string`. The AT clause supports the semicolon (;) as a field delimiter in `location-string`. If no semicolon is present, a period is the field delimiter. The use of semicolons allows file names and extensions to be used in the database and owner fields.

The string in the AT clause can also contain local or global variable names enclosed in braces ({`variable-name`}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains `'bostonase.master.dbo.{@myfunction}'` indicates that `@myfunction` is a SQL variable and that the current contents of the `@myfunction` variable should be substituted when the remote procedure is used.

A proxy function can return any data type except DECIMAL, NUMERIC, LONG VARCHAR, LONG NVARCHAR, LONG BINARY, XML, or any spatial data type.

Remarks

The CREATE FUNCTION statement creates a function in the database. A function can be created for another user by specifying an owner name. Subject to privileges, a function can be used in exactly the same way as other non-aggregate functions.

When functions are executed, not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker. Therefore, make sure to qualify all object names (tables, procedures, and so on) with their appropriate owner.

All functions are treated as deterministic unless they are declared NOT DETERMINISTIC. Deterministic functions return a consistent result for the same parameters, and are free of side effects. That is, the database server assumes that two successive calls to the same function with the same parameters returns the same result, and does not have any unwanted side effects on the query's semantics.

If a function returns a result set, it cannot also set output parameters or return a return value.

Privileges

You must have the CREATE PROCEDURE system privilege to create functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create functions owned by others.

You must also have the CREATE EXTERNAL REFERENCE system privilege to create an external function.

No privilege is required to create temporary functions.

To replace an existing function, you must be the owner of the function, or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side Effects

Automatic commit, even for temporary functions.

Standards

ANSI/ISO SQL Standard

CREATE FUNCTION is a core feature of ANSI/ISO SQL Standard, though some of its components supported in the software are optional SQL Language Features. A subset of these features includes:

- The SQL SECURITY clause is optional Language Feature T324.
- The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL function is Language Feature T041.
- The ability to create or modify a schema object within a SQL function, using statements such as CREATE TABLE or DROP TRIGGER, is Language Feature T651.
- The ability to use a dynamic-SQL statement within a SQL function, including statements such as EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is Language Feature T652.

Several clauses of the CREATE FUNCTION statement are not in the standard. These include:

- The TEMPORARY clause.
- The ON EXCEPTION RESUME clause.
- The optional DEFAULT clause for a specific routine parameter.
- The specification of a Transact-SQL function using the AS clause.
- The optional OR REPLACE clause.

Transact-SQL

CREATE FUNCTION is supported by Adaptive Server Enterprise. Adaptive Server Enterprise does not support the optional IN keyword for function parameters.

Example

The following function concatenates a firstname string and a lastname string.

```
CREATE FUNCTION fullname(
   firstname CHAR(30),
   lastname CHAR(30) )
RETURNS CHAR(61)
```

```
BEGIN
   DECLARE name CHAR(61);
   SET name = firstname || ' ' || lastname;
   RETURN (name);
END;
```

The following example replaces the fullname function created in the first example. After replacing the function, the local variable `name` is removed:

```
CREATE OR REPLACE FUNCTION fullname(
   firstname CHAR(30),
   lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
   RETURN ( firstname || ' ' || lastname );
END;
```

The following examples illustrate the use of the fullname function.

Return a full name from two supplied strings:

```
SELECT fullname ( 'joe', 'smith' );
```

fullname('joe', 'smith')
joe smith

List the names of all employees:

```
SELECT fullname ( GivenName, Surname )
FROM GROUPO.Employees;
```

fullname (GivenName, Surname)
Fran Whitney
Matthew Cobb
Philip Chin
Julie Jordan
...

The following function uses Transact-SQL syntax:

```
CREATE FUNCTION DoubleIt( @Input INT )
RETURNS INT
AS
BEGIN
  DECLARE @Result INT
  SELECT @Result = @Input * 2
  RETURN @Result
END;
```

The statement `SELECT DoubleIt( 5 );` returns a value of `10`.

The following example creates a function called fullname and sets the data types of the firstname and lastname parameters to the data types of the Surname and Givenname column of the Employees table by using a %TYPE attribute:

```
CREATE OR REPLACE FUNCTION fullname(
   firstname Employees.Surname%TYPE,
   lastname Employees.GivenName%TYPE )
RETURNS LONG VARCHAR
BEGIN
   RETURN ( firstname || ' ' || lastname );
END;
SELECT fullname ( Surname, GivenName )FROM GROUPO.Employees;
```

Related Information

Stored Procedures, Triggers, Batches, and User-defined Functions
Transact-SQL Batches
%TYPE and %ROWTYPE Attributes [page 115]
Creating a User-defined Function
ALTER FUNCTION Statement [page 682]
CREATE FUNCTION Statement [External Call] [page 866]
CREATE FUNCTION Statement [Web Service] [page 874]
BEGIN Statement [page 784]
CREATE PROCEDURE Statement [page 913]
DROP FUNCTION Statement [page 1097]
RETURN Statement [page 1343]
SQL Data Types [page 129]

1.4.4.66 CREATE FUNCTION Statement [External Call]

Creates an interface to a native or external function.

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [ owner.]function-name
( [ parameter, ... ] )
RETURNS data-type
[ SQL SECURITY { INVOKER | DEFINER } ]
[ [ NOT ] DETERMINISTIC ]
{ EXTERNAL NAME 'native-call'
   | EXTERNAL NAME 'c-call' LANGUAGE { C_ESQL32 | C_ESQL64 | C_ODBC32 |
C_ODBC64 }
   | EXTERNAL NAME 'clr-call' LANGUAGE CLR
   | EXTERNAL NAME 'perl-call' LANGUAGE PERL
   | EXTERNAL NAME 'php-call' LANGUAGE PHP
   | EXTERNAL NAME 'java-call' LANGUAGE JAVA
   | EXTERNAL NAME 'js-call' LANGUAGE JS }
```

```
parameter :
```

```
    [ IN ] parameter-name data-type [ DEFAULT expression ]

result-column :
column-name data-type

native-call :
[ system-configuration:]function-name@library-file-prefix[.{ so | dll} ]

system-configuration :
{ generic-operating-system | specific-operating-system } [ (processor-
architecture) ]

generic-operating-system :
{ UNIX | Windows }

specific-operating-system :
{ AIX | HPUX | Linux | OSX | Solaris | WindowsNT }

processor-architecture :
{ 32 | 64 | ARM | IA64 | PPC | SPARC | X86 | X86_64 }

c-call :
[ operating-system:]function-name@library; ...

operating-system :
UNIX

clr-call :
dll-name::function-name( param-type-1[, ... ] )

perl-call :
<file=perl-file> $sa_perl_return = perl-subroutine( $sa_perl_arg0[, ... ] )

php-call :
<file=php-file> print php-func( $argv[1][, ... ] )

java-call :
[package-name.]class-name.method-name java-method-signature

java-method-signature :
( [ java-field-descriptor, ... ] ) java-return-descriptor

java-field-descriptor and java-return-descriptor :
{ Z
  | B
  | S
  | I
  | J
  | F
  | D
  | C
  | V
  | [descriptor
  | Lclass-name;
```

```
}
```

```
js-call :
        <js-return-descriptor><file=js-object> js-func( js-field-
descriptor[ ...])
```

```
js-field-descriptor and js-return-descriptor :
{ S
  | B
  | I
  | U
  | D
  | [descriptor
}
```

Parameters

Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

When functions are executed, then not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, then an error is given.

OR REPLACE clause

Specifying CREATE OR REPLACE FUNCTION creates a new function, or replaces an existing function with the same name. This clause changes the definition of the function, but preserves existing privileges.

RETURNS clause

Use the RETURNS clause to specify the data type for the result of the function. The RETURNS clause must be the first clause of the statement.

SQL SECURITY clause

The SQL SECURITY clause defines whether the function is executed as the INVOKER (the user who is calling the function), or as the DEFINER (the user who owns the function). The default is DEFINER. For external calls, this clause establishes the ownership context for unqualified object references in the external environment.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the function. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following function:

```
CREATE FUNCTION user1.myFunc()
   RETURNS INT
   SQL SECURITY INVOKER
   BEGIN
     DECLARE res INT;
     SELECT COUNT(*) INTO res FROM table1;
     RETURN res;
   END;
```

If user2 attempts to run this function and a table user2.table1 *does not* exist, a table lookup error results. Additionally, if a user2.table1 *does* exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

[NOT] DETERMINISTIC clause

Use this clause to indicate whether functions are deterministic or non-deterministic. If this clause is omitted, then the deterministic behavior of the function is unspecified (the default).

If a function is declared as DETERMINISTIC, then it should return the same value every time it is invoked with the same set of parameters.

If a function is declared as NOT DETERMINISTIC, then it is not guaranteed to return the same value for the same set of parameters. A function declared as NOT DETERMINISTIC is re-evaluated each time it is called in a query. This clause is required when it is known that the function result for a given set of parameters can vary.

Also, functions that have side effects such as modifying the underlying data should be declared as NOT DETERMINISTIC. For example, a function that generates primary key values and is used in an INSERT...SELECT statement should be declared NOT DETERMINISTIC:

```
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
  DECLARE keyval INTEGER;
  UPDATE counter SET x = x + increment;
  SELECT counter.x INTO keyval FROM counter;
  RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;
```

Functions can be declared as DETERMINISTIC if they always return the same value for given input parameters.

EXTERNAL NAME native-call clause

The EXTERNAL NAME clause is not supported for TEMPORARY functions.

A function using the EXTERNAL NAME clause with no LANGUAGE attribute defines an interface to a native function written in a programming language such as C. The native function is loaded by the database server into its address space.

Because `native-call` can specify multiple sets of operating systems, processors, libraries, and functions, more-precisely specified configurations take precedence over less-precisely defined configurations. For example, `Solaris(X86_64):myfunc64@mylib.so` takes precedence over `Solaris:myfunc64@mylib.so`.

For syntaxes that support `system-configuration`, if you do not specify `system-configuration`, then it is assumed that the procedure runs on all system configurations. *UNIX* represents the following operating systems: AIX, HPUX, Linux, macOS, and Solaris. The generic term *Windows* represents all versions of the Microsoft Windows operating system.

The `specific-operating-system` and `processor-architecture` values are those operating systems and processors supported by SQL Anywhere server.

The library name (library-file-prefix) is followed by the file extension which is typically .dll on Windows and .so on UNIX and Linux. In the absence of the extension, the software appends the platform-specific default file extension for libraries. For example:

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib.dll;unix:mystring@mylib.so';
```

A simpler way to write the EXTERNAL NAME clause, using platform-specific defaults, is as follows:

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib';
```

When called, the library containing the function is loaded into the address space of the database server behavior. The native function executes as part of the database server. In this case, if the function causes a fault, then the database server shuts down. Because of this behavior, loading and executing functions in an external environment using the LANGUAGE attribute is recommended. If a function causes a fault in an external environment, then the database server continues to run.

EXTERNAL NAME c-call clause

To call a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute.

When the LANGUAGE attribute is specified, then the library containing the function is loaded by an external process and the external function executes as part of that external process. In this case, if the function causes a fault, then the database server continues to run.

```
CREATE FUNCTION ODBCinsert(
  IN ProductName CHAR(30),
  IN ProductDescription CHAR(50)
)
RETURNS INT
EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'
LANGUAGE C_ODBC32;
```

EXTERNAL NAME clr-call clause

To call a Microsoft .NET function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE CLR attribute.

A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a Microsoft .NET language such as Microsoft C# or Microsoft Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate Microsoft .NET executable).

```
CREATE FUNCTION clr_interface(
    IN p1 INT,
    IN p2 UNSIGNED SMALLINT,
    IN p3 LONG VARCHAR)
RETURNS INT
EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, string ) int'
LANGUAGE CLR;
```

EXTERNAL NAME perl-call clause

To call a Perl function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PERL attribute.

A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance).

```
CREATE FUNCTION PerlWriteToConsole( IN str LONG VARCHAR)
RETURNS INT
EXTERNAL NAME '<file=PerlConsoleExample>
    WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```

EXTERNAL NAME php-call clause

To call a PHP function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PHP attribute.

A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance).

```
CREATE FUNCTION PHPPopulateTable()
RETURNS INT
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;
```

EXTERNAL NAME java-call clause

To call a Java method in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JAVA attribute.

A Java-interfacing stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the procedure or function takes place outside the database server (that is, within a Java VM).

```
CREATE FUNCTION HelloDemo( IN name LONG VARCHAR )
RETURNS INT
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)I'
LANGUAGE JAVA;
```

The descriptors for arguments and return values from Java methods have the following meanings:

Field type	Java data type
B	byte
C	char
D	double
F	float
I	int
J	long
L `class-name`;	An instance of the class `class-name`. The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, **`java/lang/String`**.
S	short
V	void

Field type	Java data type
Z	Boolean
[	Use one for each dimension of an array.

EXTERNAL NAME js-call clause

To call a JavaScript function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JS attribute.

A JavaScript stored procedure or function behaves the same as a SQL stored procedure or function except that the code is written in JavaScript and the code execution takes place outside the database server (that is, within a Node.js executable instance).

Specify the return type of the JavaScript function at the beginning of the EXTERNAL NAME string inside angle brackets. Since JavaScript does not allow pass-by-reference for simple variables inside functions, the left bracket character ([) can proceed the S, B, I, U, or D characters to indicate that a one element array is being passed to the JavaScript stored procedure. This syntax is provided to support INOUT and OUT parameters in stored procedures.

The following is a sample function definition.

```
CREATE FUNCTION SimpleJSDemo(
    IN thousands INT,
    IN hundreds INT,
    IN tens INT,
    IN ones INT)
  RETURNS INT
  EXTERNAL NAME '<I><file=SimpleJSExample> SimpleJSFunction(IIII)'
  LANGUAGE JS;
```

The descriptors for arguments and return values from JavaScript methods have the following meanings:

Field type	JavaScript data type
S	String
B	Boolean
I	Integer
U	Unsigned integer
D	Double

Remarks

The CREATE FUNCTION statement creates a function in the database. You can create functions for other users by specifying an owner. A function is invoked as part of a SQL expression.

When referencing a temporary table from multiple functions, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

If you specify an EXTERNAL NAME clause when running on macOS 10.11 or a later version, then you must either specify the full path for the `.dylib` you need to load, or place the `.dylib` file in the `lib64` directory of the SQL Anywhere install.

Privileges

You must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges to create external functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privileges, as well as the CREATE EXTERNAL REFERENCE system privilege to create external functions owned by others.

To replace an existing function, you must own the procedure or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

CREATE FUNCTION for an external language environment is a core feature of the ANSI/ISO SQL Standard, though some of its components supported in the software are optional ANSI/ISO SQL Language Features. A subset of these features include:

- The SQL SECURITY clause is optional ANSI/ISO SQL Language Feature T324.
- The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL function is ANSI/ISO SQL Language Feature T041.
- Support for LANGUAGE JAVA is optional ANSI/ISO SQL Language Feature J621.
- The ability to create or modify a schema object within an external function, using statements such as CREATE TABLE or DROP TRIGGER, is Language Feature T653.
- The ability to use a dynamic-SQL statement within an external function, including statements such as CONNECT, EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is Language Feature T654.

Several clauses of the CREATE FUNCTION statement are not in the standard. These include:

- Support for C_ESQL32, C_ESQL64, C_ODBC32, C_ODBC64, CLR, PERL, and PHP in the LANGUAGES clause are not in the standard.
- The format of `external-call` is implementation-defined.
- The optional DEFAULT clause for a specific routine parameter is not in the standard.
- The optional OR REPLACE clause is not in the standard.

Transact-SQL

CREATE FUNCTION for an external routine is supported by Adaptive Server Enterprise. Adaptive Server Enterprise only supports LANGUAGE JAVA as the external environment (SQL Language Feature J621) for an external function.

Related Information

The JavaScript External Environment
External Environment Support
External Call Interface
The ESQL and ODBC External Environments
The CLR External Environment
The Java External Environment
The PHP External Environment
The Perl External Environment
References to Temporary Tables Within Procedures
ALTER FUNCTION Statement [page 682]
CALL Statement [page 795]
CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [Web Service] [page 874]
CREATE PROCEDURE Statement [External Call] [page 921]
DROP FUNCTION Statement [page 1097]
GRANT Statement [page 1193]
SQL Data Types [page 129]

1.4.4.67 CREATE FUNCTION Statement [Web Service]

Creates a web client function that makes an HTTP or SOAP over HTTP request.

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [ owner.]function-name ( [ parameter, ... ] )
RETURNS data-type
URL url-string
[ TYPE { http-type-spec-string | soap-type-spec-string } ]
[ HEADER header-string ]
[ CERTIFICATE certificate-string ]
[ CLIENTPORT clientport-string ]
[ PROXY proxy-string ]
[ SET protocol-option-string ]
[ SOAPHEADER soap-header-string ]
[ NAMESPACE namespace-string ]
```

```
http-type-spec-string :
HTTP[: { GET
   | POST[:MIME-type ]
   | PUT[:MIME-type ]
   | DELETE
   | HEAD
   | OPTIONS } ]
```

```
soap-type-spec-string :
SOAP[:{ RPC | DOC }
```

```
parameter :
```

```
  [ IN ] parameter-name datatype [ DEFAULT expression ]

url-string :
{ HTTP | HTTPS | HTTPS_FIPS }://[user:password@]hostname[:port][/path]

protocol-option-string : option-list [, option-list ...]

option-list :
  HTTP( http-option [ ;http-option ...] )
| SOAP( soap-option [ ;soap-option ...] )
| REDIR( redir-option [ ;redir-option ...] )

http-option :
  CHUNK={ ON | OFF | AUTO }
| EXCEPTIONS={ ON | OFF | AUTO }
| VERSION={ 1.0 | 1.1 }
| KTIMEOUT=number-of-seconds

soap-option :
  OPERATION=soap-operation-name

redir-option :
  COUNT=count
| STATUS=status-list
```

Parameters

OR REPLACE clause

Specifying CREATE OR REPLACE FUNCTION creates a new function, or replaces an existing function with the same name. This clause changes the definition of the function, but preserves existing privileges. You cannot use the OR REPLACE clause with temporary functions.

function-name

The name of the function.

parameter-name

Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type.

If a parameter has a default value, it need not be specified. Parameters with no default value must be specified.

Parameters can be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

data-type

The data type of the parameter. Set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view. However, defining the data type using a %ROWTYPE that is set to a table reference variable (`TABLE REF (table-reference-variable) %ROWTYPE`) is not allowed.

Only SOAP requests support the transmission of typed data such as FLOAT, INT, and so on. HTTP requests support the transmission of strings only, so you are limited to CHAR types.

RETURNS clause

Specify one of the following to define the return type for the SOAP or HTTP function:

- CHAR
- VARCHAR
- LONG VARCHAR
- TEXT
- NCHAR
- NVARCHAR
- LONG NVARCHAR
- NTEXT
- XML
- BINARY
- VARBINARY
- LONG BINARY

The value returned is the body of the HTTP response. No HTTP header information is included. If more information is required, such as status information, use a procedure instead of a function.

The data type does not affect how the HTTP response is processed.

URL clause

Specifies the URI of the web service. The optional user name and password parameters provide a means of supplying the credentials needed for HTTP basic authentication. HTTP basic authentication base-64 encodes the user and password information and passes it in the Authentication header of the HTTP request. When specified in this way, the user name and password are passed unencrypted, as part of the URL.

For functions of type HTTP:GET, query parameters can be specified within the URL clause in addition to being automatically generated from parameters passed to a function.

```
URL 'http://localhost/service?parm=1
```

Specifying HTTPS_FIPS forces the system to use the FIPS-certified libraries. If HTTPS_FIPS is specified, but no FIPS-certified libraries are present, libraries that are not FIPS-certified are used instead.

To use a certificate from the operating system certificate store, specify a URL beginning with **`https://`**.

TYPE clause

Specifies the format used when making the web service request. SOAP:RPC is used when SOAP is specified or no TYPE clause is included. HTTP:POST is used when HTTP is specified.

The TYPE clause allows the specification of a MIME-type for HTTP:POST and HTTP:PUT types. When HTTP:PUT is used, then a MIME-type must be specified. The `MIME-type` specification is used to set the Content-Type request header and set the mode of operation to allow only a single call parameter to populate the body of the request. Only zero or one parameter may remain when making a web service function call after parameter substitutions have been processed. Calling a web service function with a NULL value or no parameter (after substitutions) results in a request with no body and a content-length of

zero. When a MIME-type is specified then the single body parameter is sent in the request as is, so the application must ensure that the content is formatted to match the MIME-type.

Some typical MIME-types include:

- text/plain
- text/html
- text/xml

When no MIME-type is specified, parameter names and values (multiple parameters are permitted) are URL encoded within the body of the HTTP request.

The keywords for the TYPE clause have the following meanings:

'HTTP:GET'

By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the URL.

For example, the following request is produced when a client submits a request from the URL `http://localhost/WebServiceName?arg1=param1&arg2=param2`:

```
GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
// <End of Request - NO BODY>
```

'HTTP:POST'

By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the body of a POST request. URL parameters are stored in the body of the request.

For example, the following request is produced when a client submits a request from the URL `http://localhost/WebServiceName?arg1=param1&arg2=param2`:

```
POST /WebServiceName HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 19
arg1=param1&arg2=param2
// <End of Request>
```

'HTTP:PUT'

HTTP:PUT is similar to HTTP:POST, but the HTTP:PUT type does not have a default media type.

The following example demonstrates how to configure a general purpose client function that uploads data to a database server running the *%SQLANYSAMP17%*`\SQLAnywhere\HTTP\put_data.sql` sample:

```
CREATE OR REPLACE FUNCTION CPUT([data] LONG VARCHAR, resnm LONG VARCHAR,
mediatype LONG VARCHAR)
    RETURNS LONG BINARY
    URL 'http://localhost/resource/!resnm'
    TYPE 'HTTP:PUT:!mediatype';
SELECT CPUT('hello world', 'hello', 'text/plain' );
```

'HTTP:DELETE'

A web service client function can be configured to delete a resource located on a server. Specifying the media type is optional.

The following example demonstrates how to configure a general purpose client function that deletes a resource from a database server running the put_data.sql sample:

```
CREATE OR REPLACE FUNCTION CDEL(resnm LONG VARCHAR, mediatype LONG VARCHAR)
    RETURNS LONG BINARY
    URL 'http://localhost/resource/!resnm'
    TYPE 'HTTP:DELETE:!mediatype';
SELECT CDEL('hello', 'text/plain' );
```

'HTTP:HEAD'

The HEAD method is identical to a GET method but the server does not return a body. A media type can be specified.

```
CREATE OR REPLACE FUNCTION CHEAD(resnm LONG VARCHAR)
    RETURNS LONG BINARY
    URL 'http://localhost/resource/!resnm'
    TYPE 'HTTP:HEAD';
SELECT CHEAD( 'hello' );
```

'HTTP:OPTIONS'

The OPTIONS method is identical to a GET method but the server does not return a body. A media type can be specified. This method allows Cross-Origin Resource Sharing (CORS).

'SOAP:RPC'

This type sets the Content-Type header to 'text/xml'. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

'SOAP:DOC'

This type sets the Content-Type header to 'text/xml'. It is similar to the SOAP:RPC type but allows you to send richer data types. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

Specifying a MIME-type for the TYPE clause automatically sets the Content-Type header to that MIME-type.

HEADER clause

When creating HTTP web service client functions, use this clause to add, modify, or delete HTTP request header entries. The specification of headers closely resembles the format specified in RFC2616 Hypertext Transfer Protocol, HTTP/1.1, and RFC822 Standard for ARPA Internet Text Messages, including the fact that only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive.

Headers can be defined as `header-name:value-name` pairs. Each header must be delimited from its value with a colon (:) and therefore cannot contain a colon. You can define multiple headers by delimiting each pair with \n, \x0d\n, <LF> (line feed), or <CR><LF>. (carriage return followed by a line feed)

Multiple contiguous white spaces within the header are converted to a single white space.

CERTIFICATE clause

To make a secure (HTTPS) request, a client must have access to the certificate used to sign the HTTP server's certificate (or any certificate higher in the signing chain). The necessary information is specified in a string of semicolon-separated keyword=value pairs. The following keywords are available:

Keyword	Abbreviation	Description
file		The file name of the certificate or specify * to use a certificate from the operating system certificate store. Cannot be specified if either the certificate or certificate_name keyword is specified.
certificate	cert	The certificate itself. Cannot be specified if either the file or certificate_name keyword is specified.
certificate_name	cert_name	The name of a certificate stored in the database. Cannot be specified if either the file or certificate keyword is specified.
company	co	The company specified in the certificate.
unit		The company unit specified in the certificate.
name		The common name specified in the certificate.
skip_certificate_name_check		Specify ON to prevent checking the database server certificate. **i Note** Setting this option to ON is not recommended because this setting prevents the database server from fully authenticating the HTTP server.
trusted_certificate		Specify NONE to use TLS without verifying the server's certificate. Connecting without verifying the server's certificate is less secure than verifying the certificate because the client can no longer protect against a man-in-the-middle attack. However, the connection is still strongly encrypted and prevents replay attacks, which is more secure than no encryption at all. When NONE is specified, no trusted root certificate is required on the client.

Keyword	Abbreviation	Description
sni_hostname		A server can serve multiple hostnames with different certificates for each. In that case, the client can send a hostname to the server during the TLS handshake indicating which host it intends to connect to so that the server sends the correct certificate. This process is called Server Name Indication (SNI). By default, the server uses the hostname that the client is connecting to. Use the sni_hostname parameter to specify a different hostname. For example, you may want to connect to the host using an IP address, which is not supported by SNI. In that case, use the sni_hostname parameter to specify which hostname should be sent to the server. SNI is not supported when acting as a TLS/HTTPS server. Using sni_hostname as a regular TLS parameter (such as in a connection string, for example "ENC=TLS(sni_hostname=...)") is supported, but has no effect.

Certificates are required only for requests that are either directed to an HTTPS server, or can be redirected from a non-secure to a secure server. Only PEM formatted certificates are supported.

CLIENTPORT clause

Identifies the port number on which the HTTP client function communicates using TCP/IP. It is provided for and recommended only for connections through firewalls that filter "outgoing" TCP/IP connections. You can specify a single port number, ranges of port numbers, or a combination of both; for example, `CLIENTPORT '85,90-97'`.

PROXY clause

Specifies the URI of a proxy server. For use when the client must access the network through a proxy. The `proxy-string` is usually an HTTP or HTTPS url-string. This is site specific information that you usually need to obtain from your network administrator. This clause indicates that the function is to connect to the proxy server and send the request to the web service through it. For an example, the following PROXY clause sets the proxy server to `proxy.example.com`:

```
PROXY http://proxy.example.com
```

SET clause

Specifies protocol-specific behavior options for HTTP, SOAP, and REDIR (redirects). Only one SET clause is permitted. The following list describes the supported SET options. CHUNK, EXCEPTIONS, VERSION, and KTIMEOUT apply to the HTTP protocol, OPERATION applies to the SOAP protocol, and COUNT and

STATUS apply to the REDIR option. REDIR options can be included with either HTTP or SOAP protocol options.

CHUNK={ ON | OFF | AUTO }

(short form CH) This HTTP option allows you to specify whether to use chunking. Chunking allows HTTP messages to be broken up into several parts. Possible values are ON (always chunk), OFF (never chunk), and AUTO (chunk only if the contents, excluding auto-generated markup, exceeds 8196 bytes). For example, the following SET clause enables chunking:

```
SET 'HTTP(CHUNK=ON)'
```

If the CHUNK option is not specified, the default behavior is AUTO. If a chunked request fails in AUTO mode with a status of 505 `HTTP Version Not Supported`, or with 501 `Not Implemented`, or with 411 `Length Required`, the client retries the request without chunked transfer-coding.

Set the CHUNK option to OFF (never chunk) if the HTTP server does not support chunked transfer-coded requests.

Since CHUNK mode is a transfer encoding supported starting in HTTP version 1.1, setting CHUNK to ON requires that the version (VER) be set to 1.1, or not be set at all, in which case 1.1 is used as the default version.

EXCEPTIONS={ ON | OFF | AUTO }

(short form EX) This HTTP option allows you to control status code handling. The default is ON.

When set to ON or AUTO, HTTP client functions will return a response for HTTP success status codes (1XX and 2XX) and all codes will raise the exception SQLE_HTTP_REQUEST_FAILED.

```
SET 'HTTP(EXCEPTIONS=AUTO)'
```

When set to OFF, HTTP client functions will always return a response, independent of the HTTP status code. The HTTP status code will not be available.

Exceptions that are not related to the HTTP status code (for example, SQLE_UNABLE_TO_CONNECT_TO_HOST) will be raised when appropriate regardless of the EXCEPTIONS setting.

VERSION={ 1.0 | 1.1 }

(short form VER) This HTTP option allows you to specify the version of the HTTP protocol that is used for the format of the HTTP message. For example, the following SET clause sets the HTTP version to 1.1:

```
SET 'HTTP(VERSION=1.1)'
```

Possible values are 1.0 and 1.1. If VERSION is not specified:

- if CHUNK is set to ON, 1.1 is used as the HTTP version
- if CHUNK is set to OFF, 1.0 is used as the HTTP version
- if CHUNK is set to AUTO, either 1.0 or 1.1 is used, depending on whether the client is sending in CHUNK mode

KTIMEOUT=number-of-seconds

(short form KTO) This HTTP option allows you to specify the keep-alive timeout criteria, permitting a web client function to instantiate and cache a keep-alive HTTP/HTTPS connection for a period of time.

To cache an HTTP keep-alive connection, the HTTP version must be set to 1.1 and KTIMEOUT set to a non-zero value. KTIMEOUT may be useful for HTTPS connections particularly, if you notice a significant performance difference between HTTP and HTTPS connections. A database connection can only cache a single keep-alive HTTP connection. Subsequent calls to a web client function using the same URI reuse the keep-alive connection. Therefore, the executing web client call must have a URI whose scheme, destination host and port match that of the cached URI, and the HEADER clause must not specify Connection: close. When KTIMEOUT is not specified, or is set to zero, HTTP/HTTPS connections are not cached.

OPERATION=soap-operation-name

(short form OP) This SOAP option allows you to specify the name of the SOAP operation, if it is different from the name of the function you are creating. The value of OPERATION is analogous to the name of a remote function call. For example, if you wanted to create a function called accounts_login that calls a SOAP operation called login, you would specify something like the following:

```
CREATE FUNCTION accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )
   RETURNS LONG BINARY
   SET 'SOAP(OPERATION=login)'
```

If the OPERATION option is not specified, the name of the SOAP operation must match the name of the function you are creating.

COUNT=count

(short form CNT) This REDIR option allows you to control redirects. See STATUS below.

STATUS=status-list

(short form STAT) This REDIR option allows you to control redirects. HTTP response status codes such as*302 Found* and *303 See Other* are used to redirect web applications to a new URI, particularly after an HTTP POST has been performed. For example, a client request could be:

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```

The web server response could be:

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

In response, the client would send another HTTP request to the new URI. The REDIR options allow you to control the maximum number of redirections allowed and which HTTP response status codes to automatically redirect.

For example, `SET 'REDIR(COUNT=3; STATUS=301,307)'` allows a maximum limit of 3 re-directions and permits redirection for 301 and 307 statuses. If one of the other redirection status codes such as 302 or 303 is received, an error is issued (SQLE_HTTP_REQUEST_FAILED).

The default redirection limit `count` is 5. By default, an HTTP client function will automatically redirect in response to all HTTP redirection status codes (301, 302, 303, 307). To disallow all redirection status codes, use `SET 'REDIR(COUNT=0)'`. In this mode, a redirection response does not result in an error (SQLE_HTTP_REQUEST_FAILED). Instead, a result set is returned with the HTTP status and response headers. This permits a caller to conditionally reissue the request based on the URI contained in the *Location* header.

A web service function specifying a POST HTTP method, which receives a *303 See Other* status issues a redirect request using the GET HTTP method.

The *Location* header can contain either an absolute path or a relative path. The HTTP client function will handle either. The header can also include query parameters and these are forwarded to the redirected location. For example, if the header contained parameters such as the following, the subsequent GET or a POST will include these parameters.

```
Location: alternate_service?a=1&b=2
```

In the above example, the query parameters are `a=1&b=2`.

The following example shows how several option settings are combined in the same SET clause:

```
CREATE FUNCTION accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )
   RETURNS LONG BINARY
   SET 'HTTP( CHUNK=ON; VERSION=1.1 ), REDIR(COUNT=5;STATUS=302,303)'
   ...
```

The following example shows the use of short forms with uppercase and lowercase letters.

```
CREATE FUNCTION accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )
   RETURNS LONG BINARY
   SET 'HTTP( CH=ON; Ver=1.1 ), REDIR(CNT=5;Stat=302,303)'
   ...
```

SOAPHEADER clause

(SOAP format only) When declaring a SOAP web service as a function, use this clause to specify one or more SOAP request header entries. A SOAP header can be declared as a static constant, or can be dynamically set using the parameter substitution mechanism (declaring IN, OUT, or INOUT parameters for hd1, hd2, and so on). A web service function can define one or more IN mode substitution parameters, but cannot define an INOUT or OUT substitution parameter.

The following example illustrates how a client can specify the sending of several header entries using parameter substitution and receiving the response SOAP header data:

```
CREATE FUNCTION soap_client( IN hd1 LONG VARCHAR, IN hd2 LONG VARCHAR, IN hd3
LONG VARCHAR)
   RETURNS LONG BINARY
   URL 'localhost/some_endpoint'
   SOAPHEADER '!hd1!hd2!hd3';
```

NAMESPACE clause

(SOAP format only) This clause identifies the method namespace usually required for both SOAP:RPC and SOAP:DOC requests. The SOAP server handling the request uses this namespace to interpret the names of the entities in the SOAP request message body. The namespace can be obtained from the WSDL (Web Services Description Language) of the SOAP service available from the web service server. The default value is the function's URL, up to but not including the optional path component.

You can specify a variable name for `namespace-string`. If the variable is NULL, the namespace property is ignored.

Remarks

The CREATE FUNCTION statement creates a web services function in the database. A function can be created for another user by specifying an owner name.

When functions are executed, not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

Parameter values are passed as part of the request. The syntax used depends on the type of request. For HTTP:GET, the parameters are passed as part of the URL; for HTTP:POST requests, the values are placed in the body of the request. Parameters to SOAP requests are always bundled in the request body.

> **i** Note
>
> For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must have the CREATE PROCEDURE system privilege to create functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create functions owned by others.

To replace an existing function, you must own the procedure or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

Not supported by Adaptive Server Enterprise.

Example

1. The following statement creates a function named cli_test1 that returns images from the get_picture service running on localhost:

```
CREATE FUNCTION cli_test1( image LONG VARCHAR )
RETURNS LONG BINARY
URL 'http://localhost/get_picture'
TYPE 'HTTP:GET';
```

2. The following statement issues an HTTP request with the URL `http://localhost/get_picture?image=widget`:

```
 SELECT cli_test1( 'widget' );
```

3. The following statement uses a substitution parameter to allow the request URL to be passed as an input parameter. The secure HTTPS request uses a certificate stored in the database. The SET clause is used to turn off CHUNK mode transfer-encoding.

```
CREATE CERTIFICATE client_cert
FROM FILE 'C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\Certificates\\rsaroot.crt';
CREATE FUNCTION cli_test2( image LONG VARCHAR, myurl LONG VARCHAR )
RETURNS LONG BINARY
URL '!myurl'
CERTIFICATE 'certificate_name=client_cert'
TYPE 'HTTPS:GET'
SET 'HTTP(CH=OFF)'
HEADER 'ASA-ID';
```

4. The following statement issues an HTTP request with the URL `http://localhost/get_picture?image=widget`:

```
CREATE VARIABLE a_binary LONG BINARY;
SET a_binary = cli_test2( 'widget', 'https://localhost/get_picture' );
SELECT a_binary;
```

5. The following example creates a function using a variable in the NAMESPACE clause
   1. The following statements create a variable for a NAMESPACE clause:

```
CREATE VARIABLE @ns LONG VARCHAR
SET @ns = 'http://wsdl.domain.com/';
```

   2. The following statement creates a function named FtoC that uses a variable in the NAMESPACE clause:

```
CREATE FUNCTION FtoC ( IN temperature LONG VARCHAR )
RETURNS LONG BINARY
URL 'http://localhost:8082/FtoCService'
TYPE 'SOAP:DOC'
NAMESPACE @ns;
```

Related Information

HTTP and SOAP Request Structures
HTTP Request Header Management
Variables Supplied to Web Services
The Database Server as an HTTP Web Server
Web Client Application Development
%TYPE and %ROWTYPE Attributes [page 115]
Tutorial: Using a Database Server to Access a SOAP/DISH Service
Tutorial: Create a Web Server and Access it from a Web Client
SQL Data Types [page 129]
ALTER FUNCTION Statement [page 682]
CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [External Call] [page 866]
CREATE PROCEDURE Statement [page 913]
CREATE PROCEDURE Statement [Web Service] [page 931]
DROP FUNCTION Statement [page 1097]
RETURN Statement [page 1343]
remote_idle_timeout Option

1.4.4.68 CREATE INDEX Statement

Creates an index on a specified table or materialized view.

Syntax

Creating an index on a table

```
CREATE [ VIRTUAL ] [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX [ IF NOT
EXISTS ] index-name
ON [ owner.]table-name
    ( column-name [ ASC | DESC ], ...
      | function-name ( argument, [ ... ] ) AS column-name )
| [WITH NULLS [ NOT ] DISTINCT ]
[ { IN | ON } dbspace-name ]
[ FOR OLAP WORKLOAD ]
```

Creating an index on a materialized view

```
CREATE [ VIRTUAL ] [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX [ IF NOT
EXISTS ] index-name
  ON [ owner.]materialized-view-name
    ( column-name [ ASC | DESC ], ...)
| [ WITH NULLS NOT DISTINCT ]
[ { IN | ON } dbspace-name ]
[ FOR OLAP WORKLOAD ]
```

Parameters

VIRTUAL clause

The VIRTUAL keyword is primarily for use by the Index Consultant. A virtual index mimics the properties of a real physical index during the evaluation of execution plans by the Index Consultant and when the PLAN function is used. You can use virtual indexes together with the PLAN function to explore the performance impact of an index, without the often time-consuming and resource-consuming effects of creating a real index.

Virtual indexes are not visible to other connections, and are dropped when the connection is closed. Virtual indexes are not used when evaluating plans for the actual execution of queries, and so do not interfere with performance.

Virtual indexes have a limit of four columns.

Creating virtual indexes (CREATE VIRTUAL INDEX statement) is not supported inside a BEGIN PARALLEL WORK statement.

UNIQUE clause

The UNIQUE attribute ensures that there will not be two rows in the table or materialized view with identical values in all the columns in the index. If you specify UNIQUE, but do not specify WITH NULLS NOT DISTINCT, each index key must be unique or contain a NULL in at least one column. For example, two entries ('a', NULL) and ('a', NULL) are each considered unique.

If you specify UNIQUE...WITH NULLS NOT DISTINCT, then the index key must be unique regardless of the NULL values. For example, two entries ('a', NULL) and ('a', NULL) are considered equal, not unique.

There is a difference between a unique constraint and a unique index. Columns of a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a unique constraint, but not a unique index, because it can include multiple instances of NULL.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for primary keys or for columns in unique constraints. Approximate numeric data types are subject to rounding errors after arithmetic operations.

Spatial columns cannot be included in a unique index.

CLUSTERED clause

The CLUSTERED attribute causes rows to be stored in an approximate key order corresponding to the index. While the database server makes an attempt to preserve key order, total clustering is not guaranteed.

If a clustered index exists, the LOAD TABLE statement inserts rows in the order of the index key, and the INSERT statement attempts to put new rows on the same page as the one containing adjacent rows, as defined by the key order.

IF NOT EXISTS clause

When the IF NOT EXISTS attribute is specified and the named index already exists, no changes are made and an error is not returned.

ASC | DESC clause

Columns are sorted in ascending (increasing) order unless descending (DESC) is explicitly specified. An index is used for both an ascending and a descending ORDER BY, whether the index was ascending or

descending. However, if an ORDER BY is performed with mixed ascending and descending attributes, an index is used only if the index was created with the same ascending and descending attributes.

function-name

The function-name clause creates an index on a function. This clause cannot be used on declared temporary tables or materialized views.

This form of the CREATE INDEX statement is a convenience method that carries out the following operations:

1. Adds a computed column named `column-name` to the table. The column is defined with a COMPUTE clause that is the specified function, along with any specified arguments. See the COMPUTE clause of the CREATE TABLE statement for restrictions on the type of function that can be specified. The data type of the column is based on the result type of the function.
2. Populates the computed column for the existing rows in the table.
3. Creates an index on the column.
   If you drop this index, its associated computed column is not dropped.

Creating an index on a function (CREATE INDEX... ON `function-name` statement) is not supported inside a BEGIN PARALLEL WORK statement.

IN | ON clause

By default, the index is placed in the same database file as its table or materialized view. You can place the index in a separate database file by specifying a dbspace name in which to put the index. This feature is useful mainly for large databases to circumvent file size limitations, or for performance improvements that might be achieved by using multiple disk devices.

If the new index can share the physical index with an existing logical index, the IN clause is ignored.

WITH NULLS NOT DISTINCT clause

This clause can only be specified if you are declaring the index to be UNIQUE and allows you to specify that NULLs in index keys are not unique. For more information, see the UNIQUE clause.

FOR OLAP WORKLOAD clause

When you specify FOR OLAP WORKLOAD, the database server performs certain optimizations and gathers statistics on the key to help improve performance for OLAP workloads. Performance improvements are most noticeable when the optimization_workload is set to OLAP.

Remarks

Indexes can improve database performance. The database server uses physical and logical indexes. A **physical index** is the actual indexing structure as it is stored on disk. A **logical index** is a reference to a physical index. If you create an index that is identical in its physical attributes to an existing index, the database server creates a logical index that shares the existing physical index. In general, indexes created by users are considered logical indexes. The database server creates physical indexes as required to implement logical indexes, and can share the same physical index among several logical indexes.

The CREATE INDEX statement creates a sorted index on the specified columns of the named table or materialized view. Indexes are automatically used to improve the performance of queries issued to the database, and to sort queries with an ORDER BY clause. Once an index is created, it is never referenced in a SQL statement again except to validate it (VALIDATE INDEX), alter it (ALTER INDEX), delete it (DROP INDEX), or in a hint to the optimizer.

Index ownership

There is no way of specifying the index owner in the CREATE INDEX statement. Indexes are always owned by the owner of the table or materialized view.

Indexes on tables

You can create indexes on base tables, and on both local and global temporary tables.

When the CREATE INDEX statement is inside a BEGIN PARALLEL WORK statement, you can only create indexes on base tables.

Indexes on views

You can create indexes on materialized views, but not on regular views.

Index name space

The name of each index must be unique for a given table or materialized view.

Exclusive use

CREATE INDEX is prevented whenever the statement affects a table or materialized view currently being used by another connection.

Automatically created indexes

The database server automatically creates indexes for primary key, foreign key, and unique constraints. These automatically created indexes are held in the same database file as the table.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

When the CREATE INDEX statement is inside the BEGIN PARALLEL WORK statement, the granularity is at the table level. CREATE INDEX statements that create indexes on the same table are executed sequentially, even if they appear inside a BEGIN PARALLEL WORK statement.

Privileges

To create an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

To create an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

Side Effects

Automatic commit in most cases. If the auto_commit_on_create_local_temp_index option is set to Off, there is no commit before creating an index on a local temporary table. Creating an index on a function (an implicit computed column) causes a checkpoint.

Column statistics are updated (or created if they do not exist).

The CREATE INDEX statement only applies an exclusive lock to the table for a limited time at the beginning and at the end of its execution. In the middle of the CREATE INDEX statement's execution, a shared lock is applied to the table. Statements that require exclusive access to the table for the entirety of their execution cannot be executed concurrently with the CREATE INDEX statement even though its exclusive lock is only temporary. However, statements that only require shared access can be executed on the table, but only in the middle of the CREATE INDEX statement's execution.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Create a two-column index on the Employees table.

```
CREATE INDEX employee_name_index
ON GROUPO.Employees
( Surname, GivenName );
```

Create an index on the SalesOrderItems table for the ProductID column.

```
CREATE INDEX item_prod
ON GROUPO.SalesOrderItems
( ProductID );
```

Use the SORTKEY function to create an index on the Description column of the Products table, sorted according to a Russian collation. As a side effect, the statement adds a computed column desc_ru to the table. For this example to succeed, you must also have SELECT privilege on the Products table.

```
CREATE INDEX ix_desc_ru
ON GROUPO.Products (
 SORTKEY( Description, 'rusdict' )
 AS desc_ru );
```

Related Information

Advanced: Logical and Physical Indexes
Indexes
Optimize Indexes to Improve Performance
Snapshot Isolation
Clustered Indexes
Computed Columns

OLAP Support
Physical Limitations on Size and Number of Databases
Creating an Index
Obtaining Index Recommendations for Queries (Interactive SQL)
auto_commit_on_create_local_temp_index Option
DROP INDEX Statement [page 1098]
CREATE STATISTICS Statement [page 991]
optimization_workload Option

1.4.4.69 CREATE LDAP SERVER Statement

Creates an LDAP server configuration object.

Syntax

```
CREATE LDAP SERVER ldapua-server-name
[ ldapua-server-attribs ... ]
[ WITH ACTIVATE ]
```

```
ldapua-server-attribs :
SEARCH DN search-dn-attributes ...
| AUTHENTICATION URL { 'url-string' | NULL }
| CONNECTION TIMEOUT timeout-value
| CONNECTION RETRIES retry-value
| TLS { ON | OFF }
```

```
search-dn-attributes :
URL { 'url-string' | NULL }
| ACCESS ACCOUNT { 'dn-string' | NULL }
| IDENTIFIED BY ( 'password' | NULL }
| IDENTIFIED BY ENCRYPTED { encrypted-password | NULL }
```

Parameters

SEARCH DN clause

There is no default value for any parameter in the SEARCH DN clause.

URL

Use this clause to specify to identify the host (by name or by IP address), port number, and search to be performed to do the lookup of the **LDAP Distinguished Name (DN)** for a given user ID. `url-string` is validated for correct LDAP URL syntax before it is stored in ISYSLDAPSERVER. The maximum size for this string is 1024 bytes.

The format of `url-string` must comply with the LDAP URL standard. See The LDAP URL Format .

ACCESS ACCOUNT

Use this clause to specify the DN used by the database server to connect to the LDAP server. This is not a SQL Anywhere user, but a user created in the LDAP server specifically for logging in to the LDAP

server. This user must have permissions within the LDAP server to search for DNs by user ID in the locations specified in the SEARCH DN URL clause. The maximum size for this string is 1024 bytes.

IDENTIFIED BY

Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT. The maximum size is 255 bytes and cannot be set to NULL.

IDENTIFIED BY ENCRYPTED

Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT, provided in encrypted form, and is a binary value stored somewhere on disk. The maximum size of the binary is 289 bytes, and cannot be set to NULL. IDENTIFIED BY ENCRYPTED allows the password to be retrieved and used, without it becoming known.

AUTHENTICATION URL clause

Use this clause to specify the `url-string` that identifies the host by name or IP address, and the port number of the LDAP server to use to authenticate a user. The DN of the user obtained from a prior DN search and the user password are used to bind a new connection to the authentication URL. A successful connection to the LDAP server is considered proof of the identity of the connecting user. There is no default value for this parameter. For size limits to this string, see the SYSLDAPSERVER system view.

CONNECTION TIMEOUT clause

Use this clause to specify the connection timeout, in milliseconds, to the LDAP server, both for searches for the DN and for authentication. The default value is 10 seconds.

CONNECTION RETRIES clause

Use this clause to specify the number of retries for connections to the LDAP server, both for searches for the DN and for authentication. The valid range of values is 1-60. The default is 3.

TLS clause

Use this clause to specify the use of the TLS protocol on connections to the LDAP server, both for the DN searches and for authentication. The valid values are ON or OFF. The default is OFF. Use the Secure LDAP protocol by using `ldaps://` to begin the URL instead of `ldap://`. The TLS option must be set to OFF when using Secure LDAP.

WITH ACTIVATE clause

Use this clause to activate the LDAP server for immediate use. This clause permits the definition and activation of LDAP User Authentication in one statement, changing the state of the new LDAP server to READY.

Remarks

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example sets search parameters, authentication URL, 3 second timeout, and activates the LDAP server so it can begin authenticating users. A connection is made to the LDAP server without TLS or SECURE LDAP protocols. In addition to the privileges required to execute the CREATE LDAP SERVER statement, you must also have the SET ANY SECURITY system privilege to set the login_mode option in the following example.

```
SET OPTION PUBLIC.login_mode = 'Standard,LDAPUA';
CREATE LDAP SERVER apps_primary
    SEARCH DN
        URL  'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*'
        ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
        IDENTIFIED BY 'Secret99Password'
    AUTHENTICATION URL 'ldap://voyager:389/'
    CONNECTION TIMEOUT 3000
    WITH ACTIVATE;
```

This example uses the same search parameters, but specifies `ldaps://` so that a Secure LDAP connection is established with the LDAP server on host voyager, port 636. Only LDAP clients using the Secure LDAP protocol may connect on this port. The database security option Trusted_certificate_file must be set with a filename containing the certificate of the Certificate Authority (CA) that signed the certificate used by the LDAP server at `'ldaps://voyager:636'`. During the handshake with the LDAP server, the certificate presented by the LDAP server is verified by the database server to ensure that it is signed by one of the certificates listed in the file. The ACCESS ACCOUNT and IDENTIFIED BY parameters provided to the LDAP server are verified by the LDAP server as well.

```
SET OPTION PUBLIC.login_mode = 'Standard,LDAPUA';
SET OPTION PUBLIC.trusted_certificates_file = '/opt/sap/shared/trusted.txt';
CREATE LDAP SERVER secure_primary
    SEARCH DN
        URL  'ldaps://voyager:636/dc=MyCompany,dc=com??sub?cn=*'
        ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
        IDENTIFIED BY 'Secret99Password'
    AUTHENTICATION URL 'ldaps://voyager:636/'
    CONNECTION TIMEOUT 3000
    WITH ACTIVATE;
```

Related Information

ALTER LDAP SERVER Statement [page 687]
DROP LDAP SERVER Statement [page 1100]
VALIDATE LDAP SERVER Statement [page 1479]

1.4.4.70 CREATE LOGIN POLICY Statement

Creates a login policy.

Syntax

```
CREATE LOGIN POLICY policy-name policy-options

policy options :
policy-option [ policy-option ... ]

policy-option :
policy-option-name = policy-option-value

policy-option-value :
{ UNLIMITED | option-value }
```

Parameters

policy-name

The name of the login policy.

policy-option-name

The name of the login policy option.

policy-option-value

The value assigned to the login policy option. If you specify UNLIMITED, no limits are imposed.

Remarks

If you do not specify a policy option, then the corresponding root login policy option is always used. However, new policies do not inherit the max_non_dba_connections and root_auto_unlock_time policy options, these are root policy-only options.

For all unspecified settings, the new policy does not make static copies from the root login policy. Unspecified settings always default back to the root login policy. This means that a change to a root login policy option also affects all those policies for which the option was not specified.

All new databases include a root login policy. You can modify the root login policy values, but you cannot delete the policy. An overview of the default values for the root login policy is provided in the parameters section.

Privileges

You must have the MANAGE ANY LOGIN POLICY system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates the Test1 login policy. This example has an unlimited password life and allows the user a maximum of five attempts to enter a correct password before the account is locked.

```
CREATE LOGIN POLICY Test1
PASSWORD_LIFE_TIME=UNLIMITED
MAX_FAILED_LOGIN_ATTEMPTS=5;
```

The following example shows typical settings for a new login policy (ldap_user_policy) that uses LDAP user authentication. Both a primary and a secondary server configuration object (previously created) are specified to allow failover to the secondary LDAP server, and the ability to failover to standard authentication is allowed when system resources, network resources, or, both primary and secondary LDAP servers are unresponsive. This example provides a combination of authentication options that permits responsiveness with cached values when an LDAP server cannot keep up with incoming requests. This example assumes that the login_mode database option includes 'Standard'. You cannot paste and run this example since the primary and secondary servers mentioned in the example are fictitious.

```
CREATE LOGIN POLICY ldap_user_policy
LDAP_PRIMARY_SERVER=ldapsrv1
LDAP_SECONDARY_SERVER=ldapsrv2
LDAP_FAILOVER_TO_STD=ON;
```

Related Information

Login Policies
Creating a Login Policy
Assigning a Login Policy to an Existing User
ALTER LOGIN POLICY Statement [page 690]
ALTER USER Statement [page 769]
COMMENT Statement [page 808]
CREATE USER Statement [page 1045]
DROP LOGIN POLICY Statement [page 1101]
DROP USER Statement [page 1143]
Login Policy Options and Default Values

1.4.4.71 CREATE MATERIALIZED VIEW Statement

Creates a materialized view.

Syntax

```
CREATE MATERIALIZED VIEW
[ owner [,....]materialized-view-name [ ( alt-column-names, ... ) ]
[ IN dbspace-name ]
AS select-statement
[ CHECK { IMMEDIATE | MANUAL } REFRESH ]
```

```
alt-column-names :
( column-name)
```

Parameters

alt-column-names clause

Use this clause to specify alternate names for the columns in the materialized view. If you specify alternate columns names, the number of columns listed in `alt-column-names` must match the number of columns in `select-statement`. If you do not specify alternate column names, the names are set to those in `select-statement`.

IN clause

Use this clause to specify the dbspace in which to create the materialized view. If this clause is not specified, then the materialized view is created in the dbspace specified by the default_dbspace option. Otherwise, the system dbspace is used.

AS clause

Use this clause to specify, in the form of a SELECT statement, the data to use to populate the materialized view. A materialized view definition can only reference base tables; it cannot reference views, other

materialized views, or temporary tables. `select-statement` must contain column names or have an alias name specified. If you specify `alt-column-names`, those names are used instead of the aliases specified in `select-statement`.

Column names in the SELECT statement must be specified explicitly; you cannot use the `SELECT *` construct. For example, you cannot specify `CREATE MATERIALIZED VIEW matview AS SELECT * FROM table-name`. Also, you should fully qualify objects names in the `select-statement`.

CHECK clause

Use this clause to validate the statement without actually creating the view. When you specify the CHECK clause:

- The database server performs the normal language checks that would be carried out if CREATE MATERIALIZED VIEW was executed without the clause, and any errors generated are returned as usual.
- The database server does not perform the actual creation of the view, so certain errors that would occur at creation time are not generated. For example, an error indicating that the specified view name already exists is not generated. This allows you to use the CHECK clause to test intended changes to the definition of the view, without a conflict with the naming of the view.
- If CHECK IMMEDIATE REFRESH is used then the database server verifies that the syntax is valid for an immediate view and returns any errors.
- No changes are made to the database, and nothing is recorded in the transaction log.
- There is an implicit commit at the beginning of statement execution and a rollback at the end to release all locks obtained during execution.

Remarks

When you create a materialized view, it is a manual view and uninitialized. That is, it has a manual refresh type, and it has not been refreshed (populated with data). To initialize the view, execute a REFRESH MATERIALIZED VIEW statement, or use the sa_refresh_materialized_views system procedure.

You can encrypt a materialized view, change its PCTFREE setting, change its refresh type, and enable or disable its use by the optimizer. However, you must create the materialized view first, and then use the ALTER MATERIALIZED VIEW to change these settings. The default values for materialized views at creation time are:

- NOT ENCRYPTED
- ENABLE USE IN OPTIMIZATION
- PCTFREE is set according to the database page size: 200 bytes for a 4 KB page size, and 100 bytes for a 2 KB page size
- MANUAL REFRESH

Several database and server options must be in effect to create a materialized view. See *Materialized Views Restrictions*

The sa_recompile_views system procedure does not affect materialized views.

Privileges

You must have the CREATE MATERIALIZED VIEW system privilege to create materialized views owned by you. You must also be the owner of, or have SELECT privileges on, the underlying object referred to by the materialized view.

You must have the CREATE ANY MATERIALIZED VIEW or CREATE ANY OBJECT system privilege to create materialized views owned by others.

Side Effects

Automatic commit. While executing, the CREATE MATERIALIZED VIEW statement places exclusive locks, without blocking, on all tables referenced by the materialized view. If one of the referenced tables cannot be locked, the statement fails and an error is returned.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a materialized view containing confidential information about employees in the SQL Anywhere sample database. You must subsequently execute a REFRESH MATERIALIZED VIEW statement to initialize the view for use, as shown in the example.

```
CREATE MATERIALIZED VIEW EmployeeConfid2 AS
SELECT EmployeeID, Employees.DepartmentID,
    SocialSecurityNumber, Salary, ManagerID,
    Departments.DepartmentName, Departments.DepartmentHeadID
FROM GROUPO.Employees, GROUPO.Departments
WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid2;
```

Related Information

Materialized Views
Materialized Views Restrictions
Advanced: Status and Properties for Materialized Views
Additional Dbspaces Considerations
ALTER MATERIALIZED VIEW Statement [page 692]

DROP MATERIALIZED VIEW Statement [page 1103]
REFRESH MATERIALIZED VIEW Statement [page 1321]
CREATE VIEW Statement [page 1051]
sa_refresh_materialized_views System Procedure [page 1681]

1.4.4.72 CREATE MESSAGE Statement [T-SQL]

Creates a message number/message string pair. The message number can be used in PRINT and RAISERROR statements.

Syntax

```
CREATE MESSAGE message-number AS message-text

message-number : integer

message-text : string
```

Parameters

message-number

The message number of the message to add. The message number for a user-defined message must be 20000 or greater.

message-text

The text of the message to add. The maximum length is 255 bytes. PRINT and RAISERROR recognize placeholders in the message text. A single message can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow the message when the text of the message is sent to the client.

The placeholders are numbered to allow reordering of the arguments when translating a message to a language with a different grammatical structure. A placeholder for an argument appears as "%nn!": a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation mark (!), where the integer represents the position of the argument in the argument list. "%1!" is the first argument, "%2!" is the second argument, and so on.

There is no parameter corresponding to the `language` argument for sp_addmessage.

Remarks

Adds a user-defined message to the ISYSUSERMESSAGE system table for use by PRINT and RAISERROR statements.

Privileges

You must have the CREATE MESSAGE or CREATE ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

CREATE MESSAGE supplies the functionality provided by the sp_addmessage system procedure in Adaptive Server Enterprise.

Example

The following example creates a new message:

```
CREATE MESSAGE 20000 AS 'End of line reached';
```

Related Information

PRINT Statement [T-SQL] [page 1312]
RAISERROR Statement [page 1315]
DROP MESSAGE Statement [page 1104]
SYSUSERMESSAGE System View [page 1968]

1.4.4.73 CREATE MIRROR SERVER Statement

Creates or replaces a mirror server that is being used for database mirroring or read-only scale-out.

Syntax

Create a mirror server

```
CREATE [ OR REPLACE ] MIRROR SERVER mirror-server-name
AS { PRIMARY | MIRROR | ARBITER | PARTNER }
[ server-option = string [ ... ] ]
```

Create a mirror server as a copy

```
CREATE [ OR REPLACE ] MIRROR SERVER mirror-server-name
AS COPY
 { FROM SERVER parent-name [ OR SERVER server-name ] | USING AUTO PARENT }
[ server-option = string [ ... ] ]
```

```
server-option :
connection_string
logfile
preferred
state_file
```

```
parent-name :
server-name | PRIMARY
```

Parameters

OR REPLACE clause

CREATE MIRROR SERVER creates the mirror server. An error is returned if a mirror server with the specified name already exists in the database.

Specifying OR REPLACE creates a mirror server if the server does not already exist in the database, and replaces it if it does exist.

AS clause

You can specify one of the following server types:

PRIMARY

The mirror server with type PRIMARY defines a virtual or logical server, rather than an actual database server. The name of this server is the alternate server name for the database. The alternate server name can be used by applications to connect to the server currently acting as the primary server. The connection string for the server marked as PRIMARY

- Defines the connection string used by copy nodes to connect to the root node or PRIMARY parent.
- Defines the connection string used by the connection parameter NodeType PRIMARY value.

There can be only one PRIMARY server for a database.

MIRROR

The mirror server with type MIRROR defines a virtual or logical server, rather than an actual database server. The name of this server is the alternate mirror server name for the database. The alternate mirror server name can be used by applications to connect to the server currently acting as the read-only mirror. The server marked as MIRROR also defines the connection string used by the NodeType connection parameter MIRROR value. There can be only one MIRROR server for a database.

ARBITER

In a database mirroring system, the arbiter server assists in determining which of the PARTNER servers takes ownership of the database. The arbiter server must be defined with a connection string that can be used by the partner servers to connect to the arbiter. There can be only one ARBITER server for a database.

PARTNER

The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

In a database mirroring system, the partners use the connection string value to connect to each other. In a read-only scale-out system, the connection string is used by a copy-node that has this server as its parent.

Mirroring or mirroring with read-only scale-out

You must define two PARTNER servers for database mirroring, and both must have a connection string and a state file.

In a database mirroring system, servers defined as PARTNER are eligible to become the primary server and take ownership of the database.

Read-only scale-out without mirroring

You must define one PARTNER server for read-only scale-out, and it must have a connection string and no state file. This server is the root server, and runs the only copy of the database that allows both read and write operations

COPY

In a read-only scale-out system, this value specifies that the database server is a copy node. All connections to the database on this server are read-only. The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

When AS COPY is specified, then you must also specify either the FROM SERVER or USING AUTO PARENT clause.

The connection string is used by the NodeType connection parameter COPY value and it is also used by other copy nodes that have this server as their parent.

When adding copy nodes to a read-only scale-out system, you can either execute a CREATE MIRROR SERVER statement for the copy node, or have the root server define the mirror server automatically.

FROM SERVER clause

This clause can only be used when AS COPY is specified. This clause constructs a tree of servers for a scale-out system and indicates which servers the copy nodes obtain transaction log pages from.

The parent can be specified using the name of the mirror server or PRIMARY. An alternate parent for the copy node can be specified using the OR SERVER clause.

In a database mirroring system that has only two levels (partner and copy nodes), the copy nodes obtain transaction log pages from the current primary or mirror server.

A copy node determines which server to connect to by using its mirror server definition that is stored in the database. From its definition, it can locate the definition of its parent, and from its parent's definition, it can obtain the connection string to connect to the parent.

You do not have to explicitly define copy nodes for the scale-out system: you can choose to have the root node define the copy nodes when they connect.

OR SERVER clause

Use the OR SERVER clause to specify an alternate parent for the copy node.

USING AUTO PARENT clause

This clause can only be used when AS COPY is specified. This clause causes the primary server to assign a parent for this server. When you use this clause to replace an existing copy node server, the definitions for the parent and alternate parent for the copy node do not change.

server-option clause

You can specify a variable name for `server-option`.

The following options are supported:

connection_string server option

Specifies the connection string to be used to connect to the server. The connection string for a mirror server should not include a user ID or password because they are not used when one mirror server connects to another mirror server.

logfile server option

Specifies the location of the file that contains one line per request that is sent between mirror servers if database mirroring is used. This file is used only for debugging.

preferred server option

Specifies whether the server is the preferred server in the mirroring system. You can specify either YES or NO. The preferred server assumes the role of primary server whenever possible. You specify this option when defining PARTNER servers.

state_file server option

Specifies the location of the file used for maintaining state information about the mirroring system. This option is required for database mirroring. In a mirroring system, a state file must be specified for servers with type PARTNER. For arbiter servers, the location is specified as part of the command to start the server.

Remarks

This statement creates or replaces a mirror server definition; it does not change a mirror server definition. To change a mirror server definition, use the ALTER MIRROR SERVER statement.

In a database mirroring system, the mirror server type can be PRIMARY, MIRROR, ARBITER, or PARTNER, and you cannot have more than two partner servers.

In a read-only scale-out system, the mirror server type can be PRIMARY, PARTNER, or COPY.

Mirror server names for servers of type PARTNER or COPY must match the names of the database servers that are part of the mirroring system (the name used with the -n server option). This requirement allows each database server to find its own definition and that of its parent. Also, all copy node servers must have unique server names. `mirror-server-name`, `parent-name`, and `server-name` above must be 7-bit ASCII characters.

To use a copy node as the arbiter for the database it is copying in a database mirroring system, create the arbiter server with a name that does not match the server name of any of the database servers in the high availability system. In this configuration, the name of the arbiter is used as a placeholder in the mirror server definition to hold the connection string for the arbiter.

> **i** Note
>
> For parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Read-only scale-out and database mirroring each require a separate license.

Privileges

You must have the MANAGE ANY MIRROR SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a mirror server that can be used as the primary server in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_primary"
```

```
 AS PRIMARY
connection_string = 'server=scaleout_primary;host=winxp-2:6871,winxp-3:6872';
```

The following statement creates a mirror server that can be used as the mirror server in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_mirror"
AS MIRROR
connection_string = 'server=scaleout_mirror;host=winxp-2:6871,winxp-3:6872';
```

The following statement creates a mirror server that can be used as the arbiter in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_arbiter"
AS ARBITER
connection_string = 'server=scaleout_arbiter;host=winxp-4:6870';
```

The following statement creates two mirror servers that can be used as partners server in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_server1"
AS PARTNER
connection_string = 'server=scaleout_server1;HOST=winxp-2:6871'
state_file = 'c:\\server1\\server1.state';

CREATE MIRROR SERVER "scaleout_server2"
AS PARTNER
connection_string = 'server=scaleout_server2;HOST=winxp-3:6872'
state_file = 'c:\\server2\\server2.state';
```

The following statement creates a copy node that can act as the arbiter in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_child"
AS COPY FROM SERVER  "scaleout_primary"
connection_string = 'server=scaleout_child;host=winxp-5:6878';
```

The following statement defines a copy node as the arbiter for a different database mirroring system:

```
CREATE MIRROR SERVER "The Arbiter"
AS ARBITER
connection_string = 'server=scaleout_child;host=winxp-5:6878';
```

The following statement preserves the current parent if `server-name` already exists. However, it does not auto-generate a new parent.

```
CREATE OR REPLACE MIRROR SERVER "server-name" AS COPY USING AUTO PARENT;
```

Example

The following example creates a mirror server using a variable for the `connection_string` parameter.

The following statement creates a variable for a connection string:

```
CREATE VARIABLE @connstr_value LONG VARCHAR ;
SET @connstr_value = '
server=new_scaleout_primary;host=winxp-2:6871,winxp-3:6872 ' ;
```

The following statement creates a mirror server using the variable @connstr_value for the `connection_string` parameter:

```
CREATE MIRROR SERVER new_scaleout_primary AS PRIMARY
connection_string = @connstr_value ;
```

Related Information

Alphabetical List of Connection Parameters
Database Mirroring
Read-only Scale-out
How Child Copy Nodes Are Added
Troubleshooting: State Information Files of the Partners and Arbiter
Preferred Database Server in a Database Mirroring System
Automatically Assign the Parent of a Copy Node
Separately Licensed Components
SYSMIRRORSERVER System View [page 1924]
SET MIRROR OPTION Statement [page 1381]
ALTER MIRROR SERVER Statement [page 695]
COMMENT Statement [page 808]
DROP MIRROR SERVER Statement [page 1105]

1.4.4.74 CREATE MUTEX Statement

Creates or replaces a mutex (lock) that can be used to lock a resource such as a file or a procedure.

Syntax

```
CREATE [ OR REPLACE | TEMPORARY ] MUTEX [ IF NOT EXISTS ] [ owner.]mutex-name
[ SCOPE { CONNECTION | TRANSACTION } ]
```

Parameters

owner

The owner of the mutex. `owner` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

mutex-name

The name of the mutex. Specify a valid identifier in the CHAR database collation. `mutex-name` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

OR REPLACE clause

Use this clause to overwrite (update) the definition of a permanent mutex of the same name, if one exists.

If the OR REPLACE clause is specified, and a mutex with this name is in use at the time, then the statement returns an error.

You cannot use this clause with the TEMPORARY or IF NOT EXISTS clauses.

TEMPORARY clause

Use this clause to create a temporary mutex.

Do not use this clause with the OR REPLACE clause.

IF NOT EXISTS clause

Use this clause to create a mutex only if it doesn't already exist. If a mutex exists with the same name, then nothing happens and no error is returned.

You cannot use this clause with the OR REPLACE clause.

SCOPE clause

Use this clause to specify whether the mutex applies to a transaction (TRANSACTION), or the connection (CONNECTION). If the SCOPE clause is not specified, then the default behavior is CONNECTION.

Remarks

Permanent and temporary mutexes and semaphores share the same namespace; therefore, you cannot create two of these objects with the same name and owner. Use of the OR REPLACE and IF NOT EXISTS clause can inadvertently cause an error related to naming. For example, if you have a permanent mutex, and you try to create a temporary semaphore with the same name, an error is returned even if you specify IF NOT EXISTS. Similarly, if you have a temporary semaphore, and you try to replace it with a permanent semaphore with the same name by specifying OR REPLACE, an error is returned because this is equivalent to attempting to create a second object with the same name.

Permanent mutex definitions persist across database restarts. However, their state information (locked or released), does not.

A temporary mutex persists until the connection that created it is terminated, or until the mutex is dropped using a DROP MUTEX statement. If another connection is waiting for a temporary mutex and the connection that created the temporary mutex is terminated, then an error is returned to the waiting connection indicating that the mutex has been deleted.

CONNECTION scope mutexes are not automatically released other than when the connection is terminated.

Privileges

Requires the CREATE ANY MUTEX SEMAPHORE system privilege.

Side Effects

Automatic commit, but only for permanent mutexes.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a connection scope mutex called protect_my_cr_section to protect a critical section of a stored procedure.

```
CREATE MUTEX protect_my_cr_section SCOPE CONNECTION;
```

Related Information

Mutexes and Semaphores
DROP MUTEX Statement [page 1107]
LOCK MUTEX Statement [page 1265]
RELEASE MUTEX Statement [page 1329]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.75 CREATE ODATA PRODUCER Statement

Creates or replaces an OData Producer.

Syntax

```
CREATE [ OR REPLACE ] ODATA PRODUCER name [ producer-clause... ]
```

```
producer-clause:
[ ADMIN USER { NULL | user } ]
[ AUTHENTICATION [ DATABASE | USER user ] ]
[ [ NOT ] ENABLED ]
[ MODEL [ FROM ] { FILE string-or-variable [ ENCODING string ] | VALUE
{ NULL | string-or-variable [ ENCODING string ] } } ]
  [ SERVICE ] ROOT string
[ USING string ]
```

Parameters

ADMIN USER clause Specifies an administrative user that the OData Producer uses to create tables, a procedure, and an event to manage repeatable requests. The user must have the CREATE TABLE, CREATE PROCEDURE, VERIFY ODATA, and MANAGE ANY EVENT system privileges. This user cannot be the same user as specified in the AUTHENTICATION clause.

AUTHENTICATION clause

Specifies how the OData server connects to the database. The default setting, DATABASE, allows users to connect with personalized credentials. These credentials are requested using Basic HTTP authentication.

Specify USER for all users to connect using the credentials of the specified user. The specified user must have the VERIFY ODATA system privilege. USER is recommended for testing and read-only producers. See OData server security considerations for more information.

ENABLED clause Specifies whether this OData Producer is enabled or disabled. By default, OData Producers are enabled.

MODEL clause

Specifies the OData Producer service model (given in OSDL), that indicates which tables and views are exposed in the OData metadata. If you specify the FILE clause, then the value must be a variable or string of the file name that contains the OSDL data. If the file name is not fully qualified, then the path is relative to the database server's current working directory. If the file name is specified, then the file is read and the contents of the file are stored in the database and logged in the transaction log. By default, tables and views are exposed based on user privileges. Tables and views without primary keys are not exposed.

Use the ENCODING clause to specify the character set of the model data or file. The default is UTF-8.

SERVICE ROOT clause

Specifies the root of the OData service on the OData server. All resources for the OData Producer are accessed by using URIs of the following format: `scheme:host:port/path-prefix/resource-path[query-options]`.

The SERVICE ROOT clause is `/path-prefix`. The value must be specified, it cannot be NULL, and it must start with a **/**. A `path-prefix` must be a validly encoded URI path component (for example, spaces escaped as %20, % escaped as %25, and non US-ASCII characters encoded UTF-8 and escaped). Service roots cannot include the characters `:?['']#@` (neither encoded nor unencoded), and they cannot include the encoded version of the / character and should not include . and .. path segments.

USING clause

Specifies additional OData Producer options. Options are specified in a semicolon-separated list of name=value pairs.

Option	Description
AccessControlAllowOrigins { `values` | * | *none*	This parameter allows you to configure the OData Producer to return the correct headers with Cross-origin Resource Sharing (CORS) requests. The producer responds to Origin and Access-Control-Request-Method HTTP request headers when the AccessControlAllowedOrigins configuration parameter is specified. The AccessControlAllowedMethods configuration parameter can be used to limit what HTTP methods are allowed to respond to CORS requests. `values` is a comma-separated list of origins that are allowed to access the resources. * means all origins. *none* (the default) means do not enable for this producer. If an allowed origin contains one or more * characters (for example http://*.domain.com), then * characters are converted to ".*". Any . characters are escaped to "\." and the resulting allowed origin interpreted as a regular expression. Allowed origins can be more complex expressions such as https?://*.domain.[a-z]{3} that match HTTP or HTTPS, multiple subdomains, and any three-letter top-level domain (.com, .net, .org, and so on.). https?://mydomain.com is not treated as a pattern because it contains no * characters. Use http://mydomain.com,https://mydomain.com instead. This option cannot be set to none if you are using AccessControlAllowMethods.
AccessControlAllowMethods `values`	Specifies the allowed methods for Cross-Origin Resource Sharing `values` is a comma-separated list of HTTP methods that are allowed when accessing the resources. The default value is GET,POST,HEAD. Allowed values are GET, HEAD, POST, PUT, PATCH, MERGE, and DELETE. To use AccessControlAllowMethods, you must specify a value for AccessControlAllowOrigins.
ConnectionAuthExpiry = `num-seconds`	Specifies how long heavily used pooled connections are valid for. The default time is 5 minutes, the minimum time is 1 second, and the maximum time is 24 hours.

Option	Description
ConnectionPoolMaximum = `num-max-connections`	Indicates the maximum number of simultaneous connections that this OData Producer keeps open for use in the connection pool. Fewer connections might be used by the connection pool depending on the OData server load. By default, the connection pool size is limited to half of the maximum number of simultaneous connections that the database server supports.
CSRFTokenTimeout = `num-seconds-valid`	Enables CSRF token checking and specifies the number of seconds that a token is valid for. By default, this value is 0, which disables CSRF token checking. Otherwise, the number of seconds must be a valid integer value from 1 to 1800.
PageSize = `num-max-entities`	Specifies the maximum number of entities to include in a retrieve entity set response before issuing a next link. The default setting is 100.
ReadOnly = { *true* | *false* }	Indicates whether modification requests should be ignored. The default setting is *false*.
RepeatRequestForDays = { `days-number` }	Specifies how long repeatable requests are valid. This value must be an integer ranging from 1 to 31. The default setting is 2. This option is only effective when the ADMIN USER clause is not NULL.
SecureOnly = { *true* | *false* }	Indicates whether the Producer should only listen for requests on the HTTPS port. The default setting is *false*.
ServiceOperationColumnNames = { *generate* | *database* }	Specifies whether the column names in the metadata should be generated or taken from the result set columns in the database when naming the properties of the ComplexType used in the ReturnType. The default setting is *generate*.

Remarks

Creates and starts an OData Producer. The OData Producer is started only if it is enabled and the database was started on a server with the -xs ODATA database server option.

When a database is started on a database server that was started with the -xs ODATA database server option, then all enabled OData Producers are started on the database server.

Privileges

You must have the MANAGE ODATA system privilege.

If you specify the AUTHENTICATION USER clause or the ADMIN USER clause, then the specified users must have the VERIFY ODATA system privilege.

Example

The following example shows how to create an OData Producer:

```
CREATE ODATA PRODUCER OrderEntryProducer
    ADMIN USER SuperODataUser
    AUTHENTICATION USER ODataUser
    ENABLED
    MODEL FILE '../../ordermodel.osdl'
    SERVICE ROOT '/orders/'
    USING 'ConnectionPoolMaximum=10;CSRFTokenTimeout=600;PageSize=100;
      ReadOnly=false;RepeatRequestForDays=3;SecureOnly=true;
      ServiceOperationColumnNames=generate';
```

Related Information

OData Support
Network Protocol Options
ALTER ODATA PRODUCER Statement [page 699]
DROP ODATA PRODUCER Statement [page 1108]
COMMENT Statement [page 808]
-xs Database Server Option
SYSODATAPRODUCER System View [page 1929]

1.4.4.76 CREATE PROCEDURE Statement

Creates a user-defined SQL procedure in the database.

Syntax

```
CREATE [ OR REPLACE ] [ TEMPORARY ] PROCEDURE [ owner.]procedure-name
( [ parameter, ... ] )
[ RESULT ( result-column, ... ) | NO RESULT SET ]
[ SQL SECURITY { INVOKER | DEFINER } ]
[ ON EXCEPTION RESUME ]
{ compound-statement | AT location-string }
```

```
parameter :
parameter-mode parameter-name data-type [ DEFAULT expression ]
| SQLCODE
| SQLSTATE
```

```
parameter-mode :
IN
| OUT
| INOUT
```

```
result-column : column-name data-type
```

Parameters

OR REPLACE clause

Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

TEMPORARY clause

Specifying CREATE TEMPORARY PROCEDURE means that the stored procedure is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Temporary stored procedures can also be explicitly dropped. You cannot perform ALTER, GRANT, or REVOKE on them, and, unlike other stored procedures, temporary stored procedures are not recorded in the catalog or transaction log.

Temporary procedures execute with the privileges of their creator (current user), or specified owner. You can specify an owner for a temporary procedure when:

- the temporary procedure is created within a permanent stored procedure
- the owner of the temporary and permanent procedure is the same

To drop the owner of a temporary procedure, you must drop the temporary procedure first.

Temporary stored procedures can be created and dropped when connected to a read-only database, and they cannot be external procedures.

For example, the following temporary procedure drops the fictitious table called CustRank. For this example, the procedure assumes that the table name is unique and can be referenced by the procedure creator without specifying the table owner:

```
CREATE TEMPORARY PROCEDURE drop_table( IN @TableName char(128) )
BEGIN
    IF EXISTS  ( SELECT * FROM SYS.SYSTAB WHERE table_name = @TableName ) THEN
       EXECUTE IMMEDIATE 'DROP TABLE "' || @TableName || '"';
       MESSAGE 'Table "' || @TableName || '" dropped' to client;
    END IF;
END;
CALL drop_table( 'CustRank' );
```

parameter

Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of these values, parameters are INOUT by default. The keywords have the following meanings:

IN

The parameter is an expression that provides a value to the procedure.

OUT

The parameter is a variable that could be given a value by the procedure.

INOUT

The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

Set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view. However, defining the data type using a %ROWTYPE that is set to a table reference variable (`TABLE REF (table-reference-variable) %ROWTYPE` is not allowed.

When procedures are executed using the CALL statement, not all parameters need to be specified. If a default value is provided in the CREATE PROCEDURE statement, missing parameters are assigned the default values. If an argument is not provided in the CALL statement, and no default is set, an error is given.

SQLSTATE and SQLCODE are special OUT parameters that output the SQLSTATE or SQLCODE value when the procedure ends. The SQLSTATE and SQLCODE special values can be checked immediately after a procedure call to test the return status of the procedure.

The SQLSTATE and SQLCODE special values are modified by the next SQL statement. Providing SQLSTATE or SQLCODE as procedure arguments allows the return code to be stored in a variable.

Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. You cannot use the OR REPLACE clause with temporary procedures. An error is returned if the procedure being replaced is already in use. Open cursors for a connection are closed when a CREATE OR REPLACE PROCEDURE statement is executed.

RESULT clause

The RESULT clause declares the number and type of columns in the result set. The parenthesized list following the RESULT keyword defines the result column names and types. This information is returned by the Embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described.

You can define the data type of the columns in the result set using the %TYPE or %ROWTYPE attribute. Use %TYPE to set a column type to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

If a RESULT clause is specified, it must be the first clause of the statement.

Some procedures can produce more than one result set, with different numbers of columns, depending on how they are executed. For example, the following procedure returns two columns under some circumstances, and one in others.

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
    IF formal = 'n' THEN
        SELECT GivenName
        FROM GROUPO.Employees
    ELSE
        SELECT Surname, GivenName
        FROM GROUPO.Employees
    END IF
END;
```

Procedures with variable result sets must be written without a RESULT clause, or in Transact-SQL. Their use is subject to the following limitations:

Embedded SQL

You must DESCRIBE the procedure call after the cursor for the result set is opened, but before any rows are returned, to get the proper shape of result set. The CURSOR `cursor-name` clause on the DESCRIBE statement is required.

ODBC, OLE DB, ADO.NET

Variable result-set procedures can be used by applications using these interfaces. The proper description of the result sets is carried out by the driver or provider.

Open Client applications

Variable result-set procedures can be used by Open Client applications.

Web services

Web services rely on the RESULTS clause of the stored procedure to determine the number and types of the column in the result set. Web services do not support procedures that return multiple result sets, nor do they support variable result sets through the use of EXECUTE IMMEDIATE.

> **i** Note
>
> If an EXECUTE IMMEDIATE statement that includes a WITH RESULT SET ON clause is used in the procedure, and if the result set that is returned from the statement is the same as the result set that is returned from the procedure, then only the first column of the EXECUTE IMMEDIATE statement's result set is returned.

If your procedure returns only one result set, you should use a RESULT clause. The presence of this clause prevents ODBC and Open Client applications from describing the result set after a cursor is opened. However, because of a behavior change in version 17.0, ODBC no longer describes the cursor after it is

opened whether or not the RESULT clause is present. Use DescribeCursor=ALWAYS connection option to get this behavior.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure's defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the procedure definition. To avoid this problem, use column aliases in the SELECT statement that generates the result set.

NO RESULT SET clause

If a NO RESULT SET clause is specified, it must be the first clause of the statement.

Declares that no result set is returned by this procedure. This is useful when an external environment needs to know that a procedure does not return a result set.

SQL SECURITY clause

The SQL SECURITY clause defines whether the procedure is executed as the INVOKER (the user who is calling the procedure), or as the DEFINER (the user who owns the procedure). The default is DEFINER.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. When SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, make sure to qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following procedure:

```
CREATE PROCEDURE user1.myProcedure()
    RESULT( columnA INT )
    SQL SECURITY INVOKER
BEGIN
    SELECT columnA FROM table1;
END;
```

If user2 attempts to run this procedure and a table user2.table1 *does not* exist, a table lookup error results. Additionally, if a user2.table1 *does* exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

ON EXCEPTION RESUME clause

This clause enables Transact-SQL-like error handling to be used within a Watcom SQL syntax procedure.

If you use ON EXCEPTION RESUME, the procedure takes an action that depends on the setting of the on_tsql_error option. If on_tsql_error is set to Conditional (the default) the execution continues if the next statement handles the error; otherwise, it exits.

Error-handling statements include the following:

- IF
- SELECT @variable =
- CASE
- LOOP
- LEAVE
- CONTINUE
- CALL
- EXECUTE
- SIGNAL
- RESIGNAL
- DECLARE

- SET VARIABLE

You should not use explicit error handling code with an ON EXCEPTION RESUME clause.

This clause is ignored within the TRY block of a BEGIN...END statement.

compound-statement

A set of SQL statements bracketed by BEGIN and END, and separated by semicolons.

AT clause

Create a proxy stored procedure on the current database for a remote procedure specified by `location-string`. The AT clause supports the semicolon (;) as a field delimiter in `location-string`. If no semicolon is present, a period is the field delimiter. This allows file names and extensions to be used in the database and owner fields.

The string in the AT clause can also contain local or global variable names enclosed in braces ({`variable-name`}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains `'bostonase.master.dbo.{@myprocedure}'` indicates that `@myprocedure` is a SQL variable and that the current contents of the `@myprocedure` variable should be substituted when the remote procedure is used.

If a remote procedure can return a result set, even if it does not always return one, then the local procedure definition must contain a RESULT clause.

Remarks

The CREATE PROCEDURE statement creates a procedure in the database. A procedure is invoked with a CALL statement.

You can create permanent stored procedures that call external or native procedures written in a variety of programming languages.

You can use PROC as a synonym for PROCEDURE.

When referencing a temporary table from multiple procedures, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

The body of a procedure is a compound statement. The compound statement starts with a BEGIN statement and concludes with an END statement. For NewDepartment the compound statement is a single INSERT bracketed by BEGIN and END statements.

Parameters to procedures can be marked as one of IN, OUT, or INOUT. By default, parameters are INOUT parameters. All parameters to the NewDepartment procedure are IN parameters, as they are not changed by the procedure. You should set parameters to IN if they are not used to return values to the caller.

Privileges

You must have the CREATE PROCEDURE system privilege to create procedures owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT privilege to create procedures owned by others.

You do not need any privilege to create temporary procedures.

To replace an existing procedure, you must own the procedure or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side Effects

Automatic commit, even for temporary procedures.

Standards

ANSI/ISO SQL Standard

CREATE PROCEDURE is a core feature of the ANSI/ISO SQL Standard, but some of its components supported in SQL Anywhere are optional SQL language features. A subset of these features includes:

- The SQL SECURITY clause is optional ANSI/ISO SQL Language Feature T324.
- The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL procedure is ANSI/ISO SQL Language Feature T041.
- The ability to create or modify a schema object within a SQL procedure, using statements such as CREATE TABLE or DROP TRIGGER, is ANSI/ISO SQL Language Feature T651.
- The ability to use a dynamic-SQL statement within a SQL procedure, including statements such as EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is ANSI/ISO SQL Language Feature T652.

Several clauses of the CREATE PROCEDURE statement are not in the standard. These include:

- The TEMPORARY clause.
- The ON EXCEPTION RESUME clause.
- The AT clause.
- The optional DEFAULT clause for a specific routine parameter.
- The RESULT and NO RESULT SET clauses. The ANSI/ISO SQL Standard uses the RETURNS keyword.
- The optional OR REPLACE clause.

Transact-SQL

CREATE PROCEDURE is supported by Adaptive Server Enterprise.

Example

The following procedure queries the Employees table and returns salaries that are within the specified percent (`percentage`) of a specified salary (`sal`):

```
CREATE OR REPLACE PROCEDURE AverageEmployees( IN percentage NUMERIC( 5,3), IN
sal NUMERIC( 20, 3 ) )
RESULT( Department CHAR(40), GivenName person_name_t, Surname person_name_t,
Salary NUMERIC( 20, 3) )
BEGIN
    DECLARE maxS NUMERIC( 20, 3 );
    DECLARE minS NUMERIC( 20, 3 );
    IF percentage >= 1 THEN
        SET percentage = percentage / 100;
    ELSEIF percentage < 0 THEN
        SELECT 'Percentage error', 'Err','Err', -1;
        RETURN;
    END IF;
    SELECT MIN( E.Salary ), MAX( E.Salary ) INTO minS, maxS
    FROM GROUPO.Employees E;
    IF sal < minS OR sal > maxS THEN
        SELECT 'Salary out of bounds', 'Err', 'Err', -2;
        RETURN;
    END IF;
    SELECT D.DepartmentName, E.GivenName, E.Surname, E.Salary
    FROM GROUPO.Employees E JOIN Departments D ON E.DepartmentID = D.DepartmentID
    WHERE E.Salary BETWEEN sal *( 1 - percentage ) AND sal * ( 1 + percentage );
END;
```

The following procedure uses a CASE statement to classify the results of a query:

```
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
    DECLARE prod_name CHAR(20);
    SELECT name INTO prod_name FROM GROUPO.Products
    WHERE ID = product_ID;
    CASE prod_name
    WHEN 'Tee Shirt' THEN
        SET type = 'Shirt'
    WHEN 'Sweatshirt' THEN
        SET type = 'Shirt'
    WHEN 'Baseball Cap' THEN
        SET type = 'Hat'
    WHEN 'Visor' THEN
        SET type = 'Hat'
    WHEN 'Shorts' THEN
        SET type = 'Shorts'
    ELSE
        SET type = 'UNKNOWN'
    END CASE;
END;
```

The following example replaces the ProductType procedure created in the previous example. After replacing the procedure, the parameters for Tee Shirt and Sweatshirt are updated:

```
CREATE OR REPLACE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
    DECLARE prod_name CHAR(20);
    SELECT name INTO prod_name FROM GROUPO.Products
    WHERE ID = product_ID;
    CASE prod_name
    WHEN 'Tee Shirt' THEN
        SET type = 'T Shirt'
    WHEN 'Sweatshirt' THEN
```

```
        SET type = 'Long Sleeve Shirt'
    WHEN 'Baseball Cap' THEN
        SET type = 'Hat'
    WHEN 'Visor' THEN
        SET type = 'Hat'
    WHEN 'Shorts' THEN
        SET type = 'Shorts'
    ELSE
        SET type = 'UNKNOWN'
    END CASE;
END;
```

The following procedure uses a cursor and loops over the rows of the cursor to return a single value:

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
    DECLARE err_notfound EXCEPTION
    FOR SQLSTATE '02000';
    DECLARE curThisCust CURSOR FOR
        SELECT CompanyName,
             CAST(SUM(SalesOrderItems.Quantity *
             Products.UnitPrice) AS INTEGER) VALUE
        FROM GROUPO.Customers
        LEFT OUTER JOIN SalesOrders
        LEFT OUTER JOIN SalesOrderItems
        LEFT OUTER JOIN Products
        GROUP BY CompanyName;
    DECLARE ThisValue INT;
    DECLARE ThisCompany CHAR(35);
    SET TopValue = 0;
    OPEN curThisCust;
    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust
        INTO ThisCompany, ThisValue;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop;
        END IF;
        IF ThisValue > TopValue THEN
            SET TopValue = ThisValue;
            SET TopCompany = ThisCompany;
            END IF;
    END LOOP CustomerLoop;
    CLOSE curThisCust;
END;
```

The following example creates the procedure NewDepartment, which performs an INSERT into the Departments table of the SQL Anywhere sample database, creating a new department:

```
CREATE PROCEDURE NewDepartment(
   IN id INT,
   IN name CHAR(35),
   IN head_id INT )
BEGIN
   INSERT
   INTO GROUPO.Departments ( DepartmentID,
       DepartmentName, DepartmentHeadID )
   VALUES ( id, name, head_id );
END;
```

The following statement creates a procedure, DepartmentsCloseToCustomerLocation, and sets its IN parameter to the data type of the ID column in the Customers table using a %TYPE attribute:

```
CREATE OR REPLACE PROCEDURE DepartmentsCloseToCustomerLocation( IN customer_ID
Customers.ID%TYPE )
```

```
  BEGIN
    DECLARE cust_rec Customers%ROWTYPE;
    SELECT City, State, Country
    INTO cust_rec.City, cust_rec.State, cust_rec.Country
    FROM Customers
    WHERE ID = customer_ID;
    SELECT Employees.Surname, Employees.GivenName, Departments.DepartmentName
    FROM Employees JOIN Departments
    ON Departments.DepartmentHeadID = Employees.EmployeeID
    WHERE Employees.City = cust_rec.City
    AND Employees.State = cust_rec.State
    AND Employees.Country = cust_rec.Country;
  END;
CALL DepartmentsCloseToCustomerLocation(158);
```

Related Information

EXECUTE IMMEDIATE Used in Procedures, Triggers, User-defined Functions, and Batches
Result Sets
References to Temporary Tables Within Procedures
Stored Procedures, Triggers, Batches, and User-defined Functions
Remote Procedure Calls (RPCs)
%TYPE and %ROWTYPE Attributes [page 115]
Creating Remote Procedures (SQL Central)
ALTER PROCEDURE Statement [page 704]
BEGIN Statement [page 784]
CALL Statement [page 795]
CREATE FUNCTION Statement [page 861]
CREATE PROCEDURE Statement [External Call] [page 921]
CREATE PROCEDURE Statement [Web Service] [page 931]
CREATE PROCEDURE Statement [T-SQL] [page 944]
CREATE SERVER Statement [page 962]
DROP PROCEDURE Statement [page 1109]
EXECUTE IMMEDIATE Statement [SP] [page 1155]
GRANT Statement [page 1193]
SQL Data Types [page 129]
on_tsql_error Option

1.4.4.77 CREATE PROCEDURE Statement [External Call]

Creates an interface to a native or external procedure.

Syntax

```
CREATE [ OR REPLACE ] PROCEDURE [ owner.]procedure-name
     ( [ parameter[, ... ] ] )
[ RESULT( result-column [, ... ] )
```

```
    | NO RESULT SET
    | DYNAMIC RESULT SETS integer-expression ]
[ SQL SECURITY { INVOKER | DEFINER } ]
{ EXTERNAL NAME 'native-call'
  | EXTERNAL NAME 'c-call' LANGUAGE { C_ESQL32 | C_ESQL64 | C_ODBC32 |
C_ODBC64 }
  | EXTERNAL NAME 'clr-call' LANGUAGE CLR
  | EXTERNAL NAME 'perl-call' LANGUAGE PERL
  | EXTERNAL NAME 'php-call' LANGUAGE PHP
  | EXTERNAL NAME 'java-call' LANGUAGE JAVA
  | EXTERNAL NAME 'js-call' LANGUAGE JS }

parameter :
[ parameter-mode ] parameter-name data-type [ DEFAULT expression ]
| SQLCODE
| SQLSTATE

parameter-mode :
IN
| OUT
| INOUT

result-column :
column-name data-type

native-call :
[ system-configuration:]function-name@library-file-prefix[.{ so | dll} ]

system-configuration :
{ generic-operating-system | specific-operating-system } [ (processor-
architecture) ]

generic-operating-system :
{ Unix | Windows }

specific-operating-system :
{ AIX | HPUX | Linux | OSX | Solaris | WindowsNT }

processor-architecture :
{ 32 | 64 | ARM | IA64 | PPC | SPARC | X86 | X86_64 }

c-call :
[ operating-system:]function-name@library; ...

operating-system :
Unix

clr-call :
dll-name::function-name( param-type-1[, ... ] )

perl-call :
<file=perl-file> $sa_perl_return = perl-subroutine( $sa_perl_arg0[, ... ] )

php-call :
<file=php-file> print php-func( $argv[1][, ... ] )
```

```
java-call :
[package-name.]class-name.method-name java-method-signature
```

```
java-method-signature :
( [ java-field-descriptor, ... ] ) java-return-descriptor
```

```
java-field-descriptor and java-return-descriptor :
{ Z
  | B
  | S
  | I
  | J
  | F
  | D
  | C
  | V
  | [descriptor
  | Lclass-name;
}
```

```
js-call :
        <js-return-descriptor><file=js-object> js-func( js-field-
descriptor[ ...])
```

```
js-field-descriptor and js-return-descriptor :
{ S
  | B
  | I
  | U
  | D
  | [descriptor
}
```

Parameters

You can create permanent stored procedures that call external or native procedures written in a variety of programming languages. You can use PROC as a synonym for PROCEDURE.

OR REPLACE clause

Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

parameter

Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of these values, then parameters are INOUT by default. The keywords have the following meanings:

IN

The parameter is an expression that provides a value to the procedure.

OUT

The parameter is a variable that could be given a value by the procedure.

INOUT

The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

You can set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

When procedures are executed using the CALL statement, not all parameters need to be specified. If a default value is provided in the CREATE PROCEDURE statement, then missing parameters are assigned the default values. If an argument is not provided in the CALL statement, and no default is set, then an error is given.

SQLSTATE and SQLCODE are special OUT parameters that output the SQLSTATE or SQLCODE value when the procedure ends. The SQLSTATE and SQLCODE special values can be checked immediately after a procedure call to test the return status of the procedure.

The SQLSTATE and SQLCODE special values are modified by the next SQL statement. Providing SQLSTATE or SQLCODE as procedure arguments allows the return code to be stored in a variable.

Specifying OR REPLACE (CREATE OR REPLACE PROCEDURE) creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is in use.

You cannot create TEMPORARY external call procedures.

RESULT clause

The RESULT clause declares the number and type of columns in the result set. The parenthesized list following the RESULT keyword defines the result column names and types. This information is returned by the Embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described.

If a RESULT clause is specified, then it must be the first clause of the statement.

Embedded SQL (LANGUAGE C_ESQL32, LANGUAGE C_ESQL64) or ODBC (LANGUAGE C_ODBC32, LANGUAGE C_ODBC64) external procedures can return 0 or 1 result sets.

Perl, PHP (LANGUAGE PERL, LANGUAGE PHP), or JavaScript external procedures cannot return result sets. Procedures that call native functions loaded by the database server cannot return result sets.

CLR or Java (LANGUAGE CLR, LANGUAGE JAVA) external procedures can return 0, 1, or more result sets.

Some procedures produce more than one result set, with different numbers of columns, depending on how they are executed. For example, the following procedure returns two columns under some circumstances, and one in others.

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
   IF formal = 'n' THEN
      SELECT GivenName
      FROM GROUPO.Employees
   ELSE
      SELECT Surname, GivenName
      FROM Employees
   END IF
```

```
END;
```

Procedures with variable result sets must be written without a RESULT clause, or in Transact-SQL. Their use is subject to the following limitations:

Embedded SQL

You must DESCRIBE the procedure call after the cursor for the result set is opened, but before any rows are returned, to get the proper shape of result set. The CURSOR `cursor-name` clause on the DESCRIBE statement is required.

ODBC, OLE DB, ADO.NET

Variable result-set procedures can be used by applications using these interfaces. The proper description of the result sets is carried out by the driver or provider.

Open Client applications

Variable result-set procedures can be used by Open Client applications.

If your procedure returns only one result set, then use a RESULT clause. The presence of this clause prevents ODBC and Open Client applications from describing the result set after a cursor is opened. However, because of a behavior change in version 17.0, ODBC no longer describes the cursor after it is opened whether or not the RESULT clause is present. Use DescribeCursor=ALWAYS connection option to get this behavior.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure's defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the procedure definition. To avoid this problem, use column aliases in the SELECT statement that generates the result set.

NO RESULT SET clause

If a NO RESULT SET clause is specified, then it must be the first clause of the statement.

Declares that no result set is returned by this procedure. This declaration can lead to a performance improvement.

DYNAMIC RESULT SETS clause

If a DYNAMIC RESULT SETS clause is specified, it must be the first clause of the statement.

Use this clause with LANGUAGE CLR and LANGUAGE JAVA calls. The DYNAMIC RESULT SETS clause is used to specify the number of dynamic result sets that will be returned by the procedure. When a RESULT clause is specified and the DYNAMIC RESULT SETS clause is not specified, it is assumed that the number of dynamic result sets is 1. When neither the RESULT clause nor the DYNAMIC RESULT SETS clause is specified, no result set is expected and an error will result if a result set is generated.

The C_ESQL32, C_ESQL64, C_ODBC32, and C_ODBC64 external environments can also return result sets (like CLR and JAVA), but they are restricted to only one dynamic result set.

Procedures that call into Perl, PHP (LANGUAGE PERL, LANGUAGE PHP), or JavaScript external functions cannot return result sets. Procedures that call native functions loaded by the database server cannot return result sets.

SQL SECURITY clause

The SQL SECURITY clause defines whether the procedure is executed as the INVOKER (the user who is calling the procedure), or as the DEFINER (the user who owns the procedure). The default is DEFINER. For external calls, this clause establishes the ownership context for unqualified object references in the external environment.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following procedure:

```
CREATE PROCEDURE user1.myProcedure()
   RESULT( columnA INT )
   SQL SECURITY INVOKER
   BEGIN
     SELECT columnA FROM table1;
   END;
```

If user2 attempts to run this procedure and a table user2.table1 *does not* exist, a table lookup error results. Additionally, if a user2.table1 *does* exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

EXTERNAL NAME native-call clause

Because `native-call` can specify multiple sets of operating systems, processors, libraries, and functions, more-precisely specified configurations take precedence over less-precisely defined configurations. For example, `Solaris(X86_64):myfunc64@mylib.so` takes precedence over `Solaris:myfunc64@mylib.so`.

For syntaxes that support `system-configuration`, if you do not specify `system-configuration`, then it is assumed that the procedure runs on all system configurations. *Unix* represents the following operating systems: AIX, HP-UX, Linux, macOS, and Solaris. The generic term Microsoft Windows represents all versions of the Microsoft Windows operating system.

If you specify *Unix* for one of the calls, then it is assumed that the other call is for Microsoft Windows.

The `specific-operating-system` and `processor-architecture` values are those operating systems and processors supported by SQL Anywhere Server.

The library name (`library-file-prefix`) is followed by the file extension, which is typically `.dll` on Microsoft Windows and `.so` on UNIX and Linux. In the absence of the extension, the software appends the platform-specific default file extension for libraries. For example:

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
```

A simpler way to write the EXTERNAL NAME clause, using platform-specific defaults, is as follows:

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mylib';
```

When called, the library containing the function is loaded into the address space of the database server. The native function executes as part of the database server. In this case, if the function causes a fault, then the database server shuts down. Because of this behavior, loading and executing functions in an external environment using the LANGUAGE attribute is recommended. If a function causes a fault in an external environment, then the database server continues to run.

EXTERNAL NAME c-call clause

To call a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute.

When the LANGUAGE attribute is specified, then the library containing the function is loaded by an external process and the external function executes as part of that external process. In this case, if the function causes a fault, then the database server continues to run.

The following is a sample procedure definition.

```
CREATE PROCEDURE ODBCinsert(
  IN ProductName CHAR(30),
  IN ProductDescription CHAR(50)
)
NO RESULT SET
EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'
LANGUAGE C_ODBC32;
```

EXTERNAL NAME clr-call clause

To call a Microsoft .NET function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE CLR attribute.

A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a Microsoft .NET language such as Microsoft C# or Microsoft Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate Microsoft .NET executable).

```
CREATE PROCEDURE clr_interface(
    IN p1 INT,
    IN p2 UNSIGNED SMALLINT,
    OUT p3 LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, out string )'
LANGUAGE CLR;
```

EXTERNAL NAME perl-call clause

To call a Perl function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PERL attribute.

A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance).

The following is a sample procedure definition.

```
CREATE PROCEDURE PerlWriteToConsole( IN str LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME '<file=PerlConsoleExample>
    WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```

EXTERNAL NAME php-call clause

To call a PHP function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PHP attribute.

A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance).

The following is a sample procedure definition.

```
CREATE PROCEDURE PHPPopulateTable()
NO RESULT SET
```

```
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;
```

EXTERNAL NAME java-call clause

To call a Java method in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JAVA attribute.

A Java-interfacing stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the procedure or function takes place outside the database server (that is, within a Java VM).

The following is a sample procedure definition.

```
CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )
NO RESULT SET
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

The descriptors for arguments and return values from Java methods have the following meanings:

Field type	Java data type
B	byte
C	char
D	double
F	float
I	int
J	long
L `class-name`;	An instance of the class `class-name`. The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, **`java/lang/String`**.
S	short
V	void
Z	Boolean
[	Use one for each dimension of an array.

EXTERNAL NAME js-call clause

To call a JavaScript function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JS attribute.

A JavaScript stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in JavaScript and the execution of the procedure or function takes place outside the database server (that is, within a Node.js executable instance).

Specify the return type of the JavaScript function at the beginning of the EXTERNAL NAME string inside angle brackets. Since JavaScript does not allow pass-by-reference for simple variables inside functions, the left bracket character ([) can proceed the S, B, I, U, or D characters to indicate that a one element array is being passed to the JavaScript stored procedure. This syntax is provided to support INOUT and OUT parameters in stored procedures.

The following is a sample procedure definition.

```
CREATE PROCEDURE JSInOutDemo( INOUT num1 INT, OUT num2 INT )
  EXTERNAL NAME '<file=JSInOutParam> JSFunctionPlusOne([I[I)'
LANGUAGE JS;
```

The descriptors for arguments and return values from JavaScript methods have the following meanings:

Field type	JavaScript data type
S	String
B	Boolean
I	Integer
U	Unsigned integer
D	Double

Remarks

Clause order is important for the following clauses, which, when specified, must appear in the order listed here:

- Result-related clauses (RESULT, NO RESULT SET, DYNAMIC RESULT SETS)
- SQL SECURITY
- EXTERNAL NAME

The CREATE PROCEDURE statement creates a procedure in the database. You can create procedures for other users by specifying an owner. A procedure is invoked with a CALL statement.

If a stored procedure returns a result set, it cannot also set output parameters or return a return value.

When referencing a temporary table from multiple procedures, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

If you specify an EXTERNAL NAME clause when running on macOS 10.11 or a later version, then you must either specify the full path for the `.dylib` you need to load, or place the `.dylib` file in the `lib64` directory of the SQL Anywhere install.

Privileges

You must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges to create external procedures owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privileges, as well as the CREATE EXTERNAL REFERENCE system privilege to create external procedures owned by others.

To replace an existing procedure, you must own the procedure or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.

- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

CREATE PROCEDURE for an external language environment is a core feature of the ANSI/ISO SQL Standard, though some of its components supported in the software are optional SQL Language Features. A subset of these features include:

- The SQL SECURITY clause is optional Language Feature T324.
- The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to an external procedure is SQL Language Feature T041.
- The ability to create or modify a schema object within an external procedure, using statements such as CREATE TABLE or DROP TRIGGER, is SQL Language Feature T653.
- The ability to use a dynamic-SQL statement within an external procedure, including statements such as CONNECT, EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is SQL Language Feature T654.
- JAVA external procedures embody SQL Language Feature J621.

Several clauses of the CREATE PROCEDURE statement are not in the standard. These include:

- Support for C_ESQL32, C_ESQL64, C_ODBC32, C_ODBC64, CLR, PERL, and PHP in the LANGUAGES clause are not in the standard. The ANSI/ISO SQL Standard supports "C" as an `environment-name` as optional Language Feature B122.
- The format of `external-call` is implementation-defined.
- The RESULT and NO RESULT SET clauses are not in the standard. The ANSI/ISO SQL Standard uses the RETURNS clause.
- The optional DEFAULT clause for a specific routine parameter is not in the standard.
- The optional OR REPLACE clause is not in the standard.

Transact-SQL

CREATE PROCEDURE for an external routine is supported by Adaptive Server Enterprise. Adaptive Server Enterprise supports C-language and Java language external routines.

Related Information

The JavaScript External Environment
References to Temporary Tables Within Procedures
The ESQL and ODBC External Environments
External Call Interface

External Environment Support
The Perl External Environment
The CLR External Environment
The PHP External Environment
The Java External Environment
%TYPE and %ROWTYPE Attributes [page 115]
ALTER PROCEDURE Statement [page 704]
CALL Statement [page 795]
CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [External Call] [page 866]
CREATE PROCEDURE Statement [page 913]
CREATE PROCEDURE Statement [Web Service] [page 931]
CREATE PROCEDURE Statement [T-SQL] [page 944]
DROP PROCEDURE Statement [page 1109]
GRANT Statement [page 1193]
SQL Data Types [page 129]

1.4.4.78 CREATE PROCEDURE Statement [Web Service]

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

Syntax

```
CREATE [ OR REPLACE ] PROCEDURE [ owner.]procedure-name ( [ parameter, ... ] )
[ RESULT ( attribute-column-name datatype, value-column-name datatype ) ]
URL url-string
[ TYPE { http-type-spec-string | soap-type-spec-string } ]
[ HEADER header-string ]
[ CERTIFICATE certificate-string ]
[ CLIENTPORT clientport-string ]
[ PROXY proxy-string ]
[ SET protocol-option-string ]
[ SOAPHEADER soap-header-string ]
[ NAMESPACE namespace-string ]
```

```
http-type-spec-string :
HTTP[: { GET
   | POST[:MIME-type ]
   | PUT[:MIME-type ]
   | DELETE
   | HEAD
   | OPTIONS } ]
```

```
soap-type-spec-string :
SOAP[:{ RPC | DOC }
```

```
parameter :
  parameter-mode parameter-name datatype [ DEFAULT expression ]
```

```
parameter-mode :
```

```
  IN
| OUT
| INOUT

url-string :
{ HTTP | HTTPS | HTTPS_FIPS }://[user:password@]hostname[:port][/path]

protocol-option-string : option-list [, option-list ...]

option-list :
  HTTP( http-option [ ;http-option ...] )
| SOAP( soap-option [ ;soap-option ...] )
| REDIR( redir-option [ ;redir-option ...] )

http-option :
  CHUNK={ ON | OFF | AUTO }
| EXCEPTIONS={ ON | OFF | AUTO }
| VERSION={ 1.0 | 1.1 }
| KTIMEOUT=number-of-seconds

soap-option :
  OPERATION=soap-operation-name

redir-option :
  COUNT=count
| STATUS=status-list
```

Parameters

OR REPLACE clause

Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

procedure-name

The name of the procedure.

parameter-name

Parameter names must conform to the rules for other database identifiers such as column names. They must have a valid SQL data type.

If a parameter has a default value, it need not be specified. Parameters with no default value must be specified.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. OUT and INOUT parameters are only supported for SOAP procedures. If you do not specify one of these values, parameters are INOUT by default. The keywords have the following meanings:

IN

The parameter is an expression that provides a value to the procedure.

OUT

The parameter is a variable that could be given a value by the procedure.

INOUT

The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

datatype

The data type of the parameter. Set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view. However, defining the data type using a %ROWTYPE that is set to a table reference variable (`TABLE REF (table-reference-variable) %ROWTYPE` is not allowed.

Only SOAP requests support the transmission of typed data such as FLOAT, INT, and so on. HTTP requests support the transmission of strings only, so you are limited to CHAR types.

RESULT clause

The RESULT clause is required to use the procedure in a SELECT statement. The RESULT clause must return two columns. The first column contains HTTP response header, status, and response body attributes, while the second column contains the values for these attributes. The RESULT clause must specify two character data types. For example, VARCHAR or LONG VARCHAR. If the RESULT clause is not specified, the default column names are Attribute and Value and their data types are LONG VARCHAR. If you are using database created with version 17.0.1 or higher, then the result set contains a third column named Instance, of type INTEGER. If the result of calling the HTTP request returns more than one attribute with the same name, then the Instance value can be used to obtain the different attribute values.

URL clause

Specifies the URI of the web service. The optional user name and password parameters provide a means of supplying the credentials needed for HTTP basic authentication. HTTP basic authentication base-64 encodes the user and password information and passes it in the Authentication header of the HTTP request. When specified in this way, the user name and password are passed unencrypted, as part of the URL.

For procedures of type HTTP:GET, query parameters can be specified within the URL clause in addition to being automatically generated from parameters passed to a procedure.

```
URL 'http://localhost/service?parm=1
```

Specifying HTTPS_FIPS forces the system to use the FIPS-certified libraries. If HTTPS_FIPS is specified, but no FIPS-certified libraries are present, libraries that are not FIPS-certified are used instead.

To use a certificate from the operating system certificate store, specify a URL beginning with **`https://`**.

TYPE clause

Specifies the format used when making the web service request. SOAP:RPC is used when SOAP is specified or no TYPE clause is included. HTTP:POST is used when HTTP is specified.

The TYPE clause allows the specification of a MIME-type for HTTP:POST and HTTP:PUT types. When HTTP:PUT is used, then a MIME-type must be specified. The `MIME-type` specification is used to set the Content-Type request header and set the mode of operation to allow only a single call parameter to populate the body of the request. Only zero or one parameter may remain when making a web service stored procedure call after parameter substitutions have been processed. Calling a web service procedure

with a NULL value or no parameter (after substitutions) results in a request with no body and a content-length of zero. When a MIME-type is specified then the single body parameter is sent in the request as is, so the application must ensure that the content is formatted to match the MIME-type.

Some typical MIME-types include:

- text/plain
- text/html
- text/xml

When no MIME-type is specified, parameter names and values (multiple parameters are permitted) are URL encoded within the body of the HTTP request.

The keywords for the TYPE clause have the following meanings:

HTTP:GET

By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the URL.

For example, the following request is produced when a client submits a request from the URL :

```
GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
// <End of Request - NO BODY>http://localhost/WebServiceName?
arg1=param1&arg2=param2
```

HTTP:POST

By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the body of a POST request. URL parameters are stored in the body of the request.

For example, the following request is produced when a client submits a request from the URL `http://localhost/WebServiceName?arg1=param1http://localhost/WebServiceName?arg1=param1&arg2=param2`:

```
POST /WebServiceName HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 19
arg1=param1&arg2=param2
// <End of Request>
```

HTTP:PUT

HTTP:PUT is similar to HTTP:POST, but the HTTP:PUT type does not have a default media type.

The following example demonstrates how to configure a general purpose client procedure that uploads data to a database server running the *%SQLANYSAMP17%*`\SQLAnywhere\HTTP\put_data.sql` sample:

```
CREATE OR REPLACE PROCEDURE CPUT([data] LONG VARCHAR, resnm LONG VARCHAR,
mediatype LONG VARCHAR)
    URL 'http://localhost/resource/!resnm'
    TYPE 'HTTP:PUT:!mediatype';
CALL CPUT('hello world', 'hello', 'text/plain' );
```

HTTP:DELETE

A web service client procedure can be configured to delete a resource located on a server. Specifying the media type is optional.

The following example demonstrates how to configure a general purpose client procedure that deletes a resource from a database server running the put_data.sql sample:

```
CREATE OR REPLACE PROCEDURE CDEL(resnm LONG VARCHAR)
    URL 'http://localhost/resource/!resnm'
    TYPE 'HTTP:DELETE';
CALL CDEL('hello', 'text/plain' );
```

HTTP:HEAD

The HEAD method is identical to a GET method but the server does not return a body. A media type can be specified.

```
CREATE OR REPLACE PROCEDURE CHEAD(resnm LONG VARCHAR)
    URL 'http://localhost/resource/!resnm'
    TYPE 'HTTP:HEAD';
CALL CHEAD( 'hello' );
```

HTTP:OPTIONS

The OPTIONS method is identical to a GET method but the server does not return a body. A media type can be specified. This method allows Cross-Origin Resource Sharing (CORS).

SOAP:RPC

This type sets the Content-Type header to 'text/xml'. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

SOAP:DOC

This type sets the Content-Type header to 'text/xml'. It is similar to the SOAP:RPC type but allows you to send richer data types. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

Specifying a MIME-type for the TYPE clause automatically sets the Content-Type header to that MIME-type.

HEADER clause

When creating HTTP web service client procedures, use this clause to add, modify, or delete HTTP request header entries. The specification of headers closely resembles the format specified in RFC2616 Hypertext Transfer Protocol, HTTP/1.1, and RFC822 Standard for ARPA Internet Text Messages, including the fact that only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive.

Headers can be defined as `header-name:value-name` pairs. Each header must be delimited from its value with a colon (:) and therefore cannot contain a colon. You can define multiple headers by delimiting each pair with \n, \x0d\n, <LF> (line feed), or <CR><LF>. (carriage return followed by a line feed)

Multiple contiguous white spaces within the header are converted to a single white space.

CERTIFICATE clause

To make a secure (HTTPS) request, a client must have access to the root certificate of the HTTP server's certificate chain. The necessary information is specified in a string of semicolon-separated keyword=value pairs. The following keywords are available:

Keyword	Abbreviation	Description
file		The file name of the certificate or specify * to use a certificate from the operating system certificate store. Cannot be specified if either the certificate or certificate_name keyword is specified.
certificate	cert	The certificate itself. Cannot be specified if either the file or certificate_name keyword is specified.
certificate_name	cert_name	The name of a certificate stored in the database. Cannot be specified if either the file or certificate keyword is specified.
company	co	The company specified in the certificate.
unit		The company unit specified in the certificate.
name		The common name specified in the certificate.
skip_certificate_name_check		Specify YES or NO. Controls whether the client library skips the check of the server host name against the database server certificate host names.
trusted_certificate		Specify NONE to use TLS without verifying the server's certificate. Connecting without verifying the server's certificate is less secure than verifying the certificate because the client can no longer protect against a man-in-the-middle attack. However, the connection is still strongly encrypted and prevents replay attacks, which is more secure than no encryption at all. When NONE is specified, no trusted root certificate is required on the client.

Keyword	Abbreviation	Description
sni_hostname		A server can serve multiple hostnames with different certificates for each. In that case, the client can send a hostname to the server during the TLS handshake indicating which host it intends to connect to so that the server sends the correct certificate. This process is called Server Name Indication (SNI). By default, the server uses the hostname that the client is connecting to. Use the sni_hostname parameter to specify a different hostname. For example, you may want to connect to the host using an IP address, which is not supported by SNI. In that case, use the sni_hostname parameter to specify which hostname should be sent to the server.

Certificates are required only for requests that are either directed to an HTTPS server, or can be redirected from a non-secure to a secure server. Only PEM formatted certificates are supported.

CLIENTPORT clause

Identifies the port number on which the HTTP client procedure communicates using TCP/IP. It is provided for and recommended only for connections through firewalls that filter "outgoing" TCP/IP connections. You can specify a single port number, ranges of port numbers, or a combination of both; for example, `CLIENTPORT '85,90-97'`.

PROXY clause

Specifies the URI of a proxy server. For use when the client must access the network through a proxy. The `proxy-string` is usually an HTTP or HTTPS url-string. This is site specific information that you usually need to obtain from your network administrator. This clause indicates that the procedure is to connect to the proxy server and send the request to the web service through it. For an example, the following PROXY clause sets the proxy server to `proxy.example.com`:

```
PROXY http://proxy.example.com
```

SET clause

Specifies protocol-specific behavior options for HTTP, SOAP, and REDIR (redirects). Only one SET clause is permitted. The following list describes the supported SET options. CHUNK, EXCEPTIONS, VERSION, and KTIMEOUT apply to the HTTP protocol, OPERATION applies to the SOAP protocol, and COUNT and STATUS apply to the REDIR option. REDIR options can be included with either HTTP or SOAP protocol options.

CHUNK={ ON | OFF | AUTO }

(short form CH) This HTTP option allows you to specify whether to use chunking. Chunking allows HTTP messages to be broken up into several parts. Possible values are ON (always chunk), OFF (never

chunk), and AUTO (chunk only if the contents, excluding auto-generated markup, exceeds 8196 bytes). For example, the following SET clause enables chunking:

```
SET 'HTTP(CHUNK=ON)'
```

If the CHUNK option is not specified, the default behavior is AUTO. If a chunked request fails in AUTO mode with a status of 505 `HTTP Version Not Supported`, or with 501 `Not Implemented`, or with 411 `Length Required`, the client retries the request without chunked transfer-coding.

Set the CHUNK option to OFF (never chunk) if the HTTP server does not support chunked transfer-coded requests.

Since CHUNK mode is a transfer encoding supported starting in HTTP version 1.1, setting CHUNK to ON requires that the version (VER) be set to 1.1, or not be set at all, in which case 1.1 is used as the default version.

EXCEPTIONS={ ON | OFF | AUTO }

(short form EX) This HTTP option allows you to control status code handling. The default is ON.

When set to ON or AUTO, HTTP client procedures will return a result set for HTTP success status codes (1XX and 2XX) and all codes will raise the exception SQLE_HTTP_REQUEST_FAILED.

```
SET 'HTTP(EXCEPTIONS=AUTO)'
```

When set to OFF, HTTP client procedures will always return a result set, independent of the HTTP status code. The result row with the word *Status* in the attribute column contains the HTTP status code in the value column.

Exceptions that are not related to the HTTP status code (for example, SQLE_UNABLE_TO_CONNECT_TO_HOST) will be raised when appropriate regardless of the EXCEPTIONS setting.

VERSION={ 1.0 | 1.1 }

(short form VER) This HTTP option allows you to specify the version of the HTTP protocol that is used for the format of the HTTP message. For example, the following SET clause sets the HTTP version to 1.1:

```
SET 'HTTP(VERSION=1.1)'
```

Possible values are 1.0 and 1.1. If VERSION is not specified:

- if CHUNK is set to ON, 1.1 is used as the HTTP version
- if CHUNK is set to OFF, 1.0 is used as the HTTP version
- if CHUNK is set to AUTO, either 1.0 or 1.1 is used, depending on whether the client is sending in CHUNK mode

KTIMEOUT=number-of-seconds

(short form KTO) This HTTP option allows you to specify the keep-alive timeout criteria, permitting a web client procedure to instantiate and cache a keep-alive HTTP/HTTPS connection for a period of time. To cache an HTTP keep-alive connection, the HTTP version must be set to 1.1 and KTIMEOUT set to a non-zero value. KTIMEOUT may be useful for HTTPS connections particularly, if you notice a significant performance difference between HTTP and HTTPS connections. A database connection can only cache a single keep-alive HTTP connection. Subsequent calls to a web client procedure using the same URI reuse the keep-alive connection. Therefore, the executing web client call must have a URI

whose scheme, destination host and port match that of the cached URI, and the HEADER clause must not specify Connection: close. When KTIMEOUT is not specified, or is set to zero, HTTP/HTTPS connections are not cached.

OPERATION=soap-operation-name

(short form OP) This SOAP option allows you to specify the name of the SOAP operation, if it is different from the name of the procedure you are creating. The value of OPERATION is analogous to the name of a remote procedure call. For example, if you wanted to create a procedure called accounts_login that calls a SOAP operation called login, you would specify something like the following:

```
CREATE PROCEDURE accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )
   SET 'SOAP(OPERATION=login)'
```

If the OPERATION option is not specified, the name of the SOAP operation must match the name of the procedure you are creating.

COUNT=count

(short form CNT) This REDIR option allows you to control redirects. See STATUS below.

STATUS=status-list

(short form STAT) This REDIR option allows you to control redirects. HTTP response status codes such as*302 Found* and *303 See Other* are used to redirect web applications to a new URI, particularly after an HTTP POST has been performed. For example, a client request could be:

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```

The web server response could be:

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

In response, the client would send another HTTP request to the new URI. The REDIR options allow you to control the maximum number of redirections allowed and which HTTP response status codes to automatically redirect.

For example, `SET 'REDIR(COUNT=3; STATUS=301,307)'` allows a maximum limit of 3 re-directions and permits redirection for 301 and 307 statuses. If one of the other redirection status codes such as 302 or 303 is received, an error is issued (SQLE_HTTP_REQUEST_FAILED).

The default redirection limit `count` is 5. By default, an HTTP client procedure will automatically redirect in response to all HTTP redirection status codes (301, 302, 303, 307). To disallow all redirection status codes, use `SET 'REDIR(COUNT=0)'`. In this mode, a redirection response does not result in an error (SQLE_HTTP_REQUEST_FAILED). Instead, a result set is returned with the HTTP status and response headers. This permits a caller to conditionally reissue the request based on the URI contained in the *Location* header.

A web service procedure specifying a POST HTTP method, which receives a *303 See Other* status issues a redirect request using the GET HTTP method.

The *Location* header can contain either an absolute path or a relative path. The HTTP client procedure will handle either. The header can also include query parameters and these are forwarded to the

redirected location. For example, if the header contained parameters such as the following, the subsequent GET or a POST will include these parameters.

```
Location: alternate_service?a=1&b=2
```

In the above example, the query parameters are `a=1&b=2`.

The following example shows how several option settings are combined in the same SET clause:

```
CREATE PROCEDURE accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )
   SET 'HTTP( CHUNK=ON; VERSION=1.1 ), REDIR(COUNT=5;STATUS=302,303)'
   ...
```

The following example shows the use of short forms with uppercase and lowercase letters.

```
CREATE PROCEDURE accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )
   SET 'HTTP( CH=ON; Ver=1.1 ), REDIR(CNT=5;Stat=302,303)'
   ...
```

SOAPHEADER clause

(SOAP format only) When declaring a SOAP web service as a procedure, use this clause to specify one or more SOAP request header entries. A SOAP header can be declared as a static constant, or can be dynamically set using the parameter substitution mechanism (declaring IN, OUT, or INOUT parameters for hd1, hd2, and so on). A web service procedure can define one or more IN mode substitution parameters, and a single INOUT or OUT substitution parameter.

The following example illustrates how a client can specify the sending of several header entries using parameter substitution and receiving the response SOAP header data:

```
CREATE PROCEDURE soap_client(INOUT hd1 LONG VARCHAR, IN hd2 LONG VARCHAR, IN
hd3 LONG VARCHAR)
   URL 'localhost/some_endpoint'
   SOAPHEADER '!hd1!hd2!hd3';
```

NAMESPACE clause

(SOAP format only) This clause identifies the method namespace usually required for both SOAP:RPC and SOAP:DOC requests. The SOAP server handling the request uses this namespace to interpret the names of the entities in the SOAP request message body. The namespace can be obtained from the WSDL (Web Services Description Language) of the SOAP service available from the web service server. The default value is the procedure's URL, up to but not including the optional path component.

You can specify a variable name for `namespace-string`. If the variable is NULL, the namespace property is ignored.

Remarks

Parameter values are passed as part of the request. The syntax used depends on the type of request. For HTTP:GET, the parameters are passed as part of the URL; for HTTP:POST requests, the values are placed in the body of the request. Parameters to SOAP requests are always bundled in the request body.

You can create or replace a web services client procedure. You can use PROC as a synonym for PROCEDURE.

For SOAP requests, the procedure name is used as the SOAP operation name by default. For more information, see the SET clause.

You cannot create TEMPORARY web services procedures.

For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the CREATE PROCEDURE system privilege to create procedures owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create procedures owned by others.

To replace an existing procedure, you must own the procedure or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

Not supported by Adaptive Server Enterprise.

Example

1. The following example creates a web service client procedure named FtoC.

```
CREATE PROCEDURE FtoC( IN temperature FLOAT,
    INOUT inoutheader LONG VARCHAR,
    IN inheader LONG VARCHAR )
  URL 'http://localhost:8082/FtoCService'
```

```
  TYPE 'SOAP:DOC'
  SOAPHEADER '!inoutheader!inheader';
```

2. The following example creates a secure web service client procedure named SecureSendWithMimeType that uses a certificate stored in the database.

```
CREATE CERTIFICATE client_cert
FROM FILE 'C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\Certificates\\rsaroot.crt';
CREATE PROCEDURE SecureSendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
CERTIFICATE 'certificate_name=client_cert'
TYPE 'HTTPS:POST:!mimeType';
CALL SecureSendWithMimeType('<hello>this is xml</hello>',
    'text/xml',
    'https://localhost:4043/EchoService'
);
```

3. The following example creates a procedure named SecureSendWithMimeType that uses a certificate from the operating system certificate store:

```
CREATE PROCEDURE SecureSendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
CERTIFICATE 'file=*'
TYPE 'HTTPS:POST:!mimeType';
```

4. The following example creates a procedure named SecureSendWithMimeType that verifies that the certificate myrootcert.crt is at the root of the database server's certificate's signing chain, but does no other checking:

```
CREATE PROCEDURE SecureSendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
CERTIFICATE 'file=myrootcert.crt;skip_certificate_name_check=ON'
TYPE 'HTTPS:POST:!mimeType';
```

5. The following example creates a procedure using a variable in the NAMESPACE clause.
   1. The following statements create a variable for a NAMESPACE clause:

```
CREATE VARIABLE @ns LONG VARCHAR
SET @ns = 'http://wsdl.domain.com/';
```

   2. The following statement creates a procedure named FtoC that uses a variable in the NAMESPACE clause:

```
CREATE PROCEDURE FtoC( IN temperature FLOAT,
    INOUT inoutheader LONG VARCHAR,
    IN inheader LONG VARCHAR )
  URL 'http://localhost:8082/FtoCService'
  TYPE 'SOAP:DOC'
  SOAPHEADER '!inoutheader!inheader'
  NAMESPACE @ns;
```

6. The following statement causes a POST request to the URL 'http://localhost/post_data' with the body of the request equal to the json array '[0,1,2]' and the Content-Type of the request set to 'application/json'.

```
CREATE OR REPLACE PROCEDURE CPOST ( [data] LONG VARCHAR, [url] LONG VARCHAR,
mediatype LONG VARCHAR )
URL '!url'
TYPE 'HTTP:POST:!mediatype';
CALL CPOST( '[0,1,2]', 'http://localhost/post_data', 'application/json' );
```

7. The following example creates a TLS connection that does not verify the server's certificate:

```
CREATE PROCEDURE myWebProc( ) URL 'HTTPS://myHost/myService' CERTIFICATE
'cert=none';
```

Related Information

SOAP Structured Data Types
HTTP and SOAP Request Structures
HTTP Request Header Management
Variables Supplied to Web Services
The Database Server as an HTTP Web Server
Web Client Application Development
Substitution Parameters Used for Clause Values
Web Client Application Development
%TYPE and %ROWTYPE Attributes [page 115]
Tutorial: Using a Database Server to Access a SOAP/DISH Service
Tutorial: Create a Web Server and Access it from a Web Client
ALTER PROCEDURE Statement [page 704]
CALL Statement [page 795]
CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [Web Service] [page 874]
CREATE PROCEDURE Statement [page 913]
CREATE PROCEDURE Statement [T-SQL] [page 944]
CREATE PROCEDURE Statement [External Call] [page 921]
DROP PROCEDURE Statement [page 1109]
GRANT Statement [page 1193]
remote_idle_timeout Option
ClientPort (CPORT) Protocol Option (Client Side Only)
HTTP request failed. Status code '%1'

1.4.4.79 CREATE PROCEDURE Statement [T-SQL]

Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

Syntax

The following subset of the Transact-SQL CREATE PROCEDURE statement is supported in SQL Anywhere.

```
CREATE [ OR REPLACE ]PROCEDURE [owner.]procedure-name
[ NO RESULT SET ]
[ [ ( ] @parameter-name data-type [ = default ] [ OUTPUT ], ... [ ) ] ]
[ WITH RECOMPILE ] AS statement-list
```

Parameters

OR REPLACE clause

Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

NO RESULT SET clause

Declares that no result set is returned by this procedure. This is useful when an external environment needs to know that a procedure does not return a result set.

WITH RECOMPILE clause

This clause is accepted for Transact-SQL compatibility, but is ignored. SQL Anywhere always recompiles procedures the first time they are executed after a database is started, and stores the compiled procedure until the database is stopped.

Remarks

The following differences between Transact-SQL and SQL Anywhere statements (Watcom SQL) are listed to help those writing in both dialects:

Variable names prefixed by @

The @ sign denotes a Transact-SQL variable name, while Watcom SQL variables can be any valid identifier, and the @ prefix is optional.

Input and output parameters

Watcom SQL procedure parameters are INOUT by default or can be specified as IN, OUT, or INOUT. Transact-SQL procedure parameters are INPUT parameters by default. They can be specified as input/output with the addition of the OUTPUT keyword. There are no output-only parameters in the Transact-SQL dialect.

When you use the Watcom SQL dialect to declare a parameter OUT, it is output-only. The mixing of dialects is not recommended because it can cause problems when the procedure declaration is unloaded and used

to rebuild the database. If the procedure declaration is unloaded and used to rebuild the database, the rebuilt procedure declaration is in the Transact-SQL dialect, the OUTPUT keyword is used, and the parameter is input/output.

Parameter default values

Watcom SQL procedure parameters are given a default value using the keyword DEFAULT, while Transact-SQL uses an equality sign (=) to provide the default value.

Returning result sets

Watcom SQL uses a RESULT clause to specify returned result sets. In Transact-SQL procedures, the column names or alias names of the first query are returned to the calling environment.

The following Transact-SQL procedure illustrates how result sets are returned from Transact-SQL stored procedures:

```
CREATE PROCEDURE showdept @deptname varchar(30)
AS
   SELECT Employees.Surname, Employees.GivenName
   FROM Departments, Employees
   WHERE Departments.DepartmentName = @deptname
   AND Departments.DepartmentID = Employees.DepartmentID;
```

The following is the corresponding Watcom SQL procedure:

```
CREATE PROCEDURE showdept2(in deptname
         varchar(30) )
RESULT ( lastname char(20), firstname char(20))
ON EXCEPTION RESUME
BEGIN
   SELECT Employees.Surname, Employees.GivenName
   FROM Departments, Employees
   WHERE Departments.DepartmentName = deptname
   AND Departments.DepartmentID = Employees.DepartmentID
END;
```

Procedure body

The body of a Transact-SQL procedure is a list of Transact-SQL statements prefixed by the AS keyword. The body of a Watcom SQL procedure is a compound statement, bracketed by BEGIN and END keywords.

Privileges

You must have the CREATE PROCEDURE privilege to create procedures owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT privilege to create procedures owned by others.

To replace an existing procedure, you must own the procedure or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

SQL Anywhere supports a subset of the Adaptive Server Enterprise CREATE PROCEDURE statement syntax.

Only Transact-SQL SQL procedures are supported in the SQL Anywhere Transact-SQL dialect. To create an external procedure you must use Watcom SQL syntax. Adaptive Server Enterprise does not support the NO RESULT SET clause. If the Transact-SQL WITH RECOMPILE optional clause is supplied, it is ignored. SQL Anywhere always recompiles procedures the first time they are executed after a database is started, and stores the compiled procedure until the database is stopped.

Groups of Transact-SQL procedures are not supported in SQL Anywhere.

Related Information

CREATE FUNCTION Statement [page 861]
CREATE PROCEDURE Statement [page 913]

1.4.4.80 CREATE PUBLICATION Statement [MobiLink] [SQL Remote]

Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

Syntax

MobiLink general use

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ]  publication-name
( article-definition, ... )
```

```
article-definition :
  TABLE table-name [ ( column-name, ... ) ]
[ WHERE search-condition ]
```

MobiLink scripted upload

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
WITH SCRIPTED UPLOAD
( article-definition, ... )
```

```
article-definition :
  TABLE table-name [ ( column-name, ... ) ]
[ USING ( [ PROCEDURE ] [ owner.] procedure-name
   FOR UPLOAD { INSERT | DELETE | UPDATE }, ... ) ]
```

MobiLink download-only publications

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner.] publication-name
FOR DOWNLOAD ONLY
( article-definition, ... )
```

```
article-definition :  TABLE table-name [ ( column-name, ... ) ]
```

SQL Remote

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner.] publication-name
( article-definition, ... )
```

```
article-definition :
  TABLE table-name [ ( column-name, ... ) ]
[ WHERE search-condition ]
[ SUBSCRIBE BY expression ]
```

Parameters

IF NOT EXISTS clause

When the IF NOT EXISTS clause is specified and the named publication already exists, no changes are made and an error is not returned.

article-definition

Publications are built from articles. Each article identifies the rows and columns of a single table that are included in the publication. A publication may not contain two articles that refer to the same table.

If a list of column-names is included in an article, only those columns are included in the publication. If no column-names are listed, all columns in the table are include in the publication. For MobiLink synchronization, if column-names are listed then all columns in the primary key of the table must be included in the list.

In the MobiLink scripted upload syntax, which is used for publications that perform scripted uploads, the article description also registers the scripts that are used to define the upload.

In the MobiLink download-only publications, which is used for download-only publications, the article specifies only the tables and columns to be downloaded.

WHERE clause

The WHERE clause lets you define the subset of rows in a table to be included in an article.

In MobiLink applications, the WHERE clause affects the rows included in the upload. (The download is defined by the download_cursor script.) In MobiLink SQL Anywhere remote databases, the WHERE clause can only refer to columns included in the article, and cannot contain subqueries, variables, or non-deterministic functions.

SUBSCRIBE BY clause

In SQL Remote, one way of defining a subset of rows of a table to be included in an article is to use a SUBSCRIBE BY clause. This clause allows many different subscribers to receive different rows from a table in a single publication definition.

Remarks

The CREATE PUBLICATION statement creates a publication in the database. A publication can be created for another user by specifying an owner name.

In MobiLink, publications are required in SQL Anywhere remote databases, and are optional in UltraLite databases. These publications and the subscriptions to them determine which data is uploaded to the MobiLink server.

You set options for a MobiLink publication with the ADD OPTION clause in the CREATE SYNCHRONIZATION SUBSCRIPTION statement or ALTER SYNCHRONIZATION SUBSCRIPTION statement.

The MobiLink scripted upload syntax creates a publication for scripted uploads. Use the USING clause to register the stored procedures that you want to use to define the upload. For each table, you can use up to three stored procedures: one each for inserts, deletes, and updates.

The MobiLink download-only publications syntax creates a download-only publication that can be synchronized with no transaction log file. When download-only publications are synchronized, downloaded rows may overwrite changes that were made to those rows in the remote database.

In SQL Remote, publishing is a two-way operation, as data can be entered at both consolidated and remote databases. In a SQL Remote installation, any consolidated database and all remote databases must have the same publication defined. Running the SQL Remote Extraction utility from a consolidated database automatically executes the correct CREATE PUBLICATION statement in the remote database.

For all syntaxes, you must have exclusive access to all tables referred to in the statement to execute the statement.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement publishes all columns and rows of two tables.

```
CREATE PUBLICATION pub_contact (
   TABLE GROUPO.Contacts,
   TABLE GROUPO.Customers
);
```

The following statement publishes only some columns of one table.

```
CREATE PUBLICATION pub_customer (
   TABLE GROUPO.Customers ( ID, CompanyName, City )
);
```

The following statement publishes only the rows for customer located in New York (NY) by including a WHERE clause that tests the State column of the Customers table.

```
CREATE PUBLICATION pub_customer (
   TABLE GROUPO.Customers ( ID, CompanyName, City, State, Status )
   WHERE State = 'NY'
);
```

The following statement publishes only some rows by providing a subscribe-by value. This method can be used only with SQL Remote.

```
CREATE PUBLICATION pub_customer (
   TABLE GROUPO.Customers ( ID, CompanyName, City, State )
   SUBSCRIBE BY State
);
```

The subscribe-by value is used as follows when you create a SQL Remote subscription.

```
CREATE SUBSCRIPTION TO pub_customer ( 'NY' )
   FOR jsmith;
```

The following example creates a MobiLink publication that uses scripted uploads:

```
CREATE PUBLICATION pub WITH SCRIPTED UPLOAD (
      GROUPO.TABLE t1 (a, b, c) USING (
         PROCEDURE my.t1_ui FOR UPLOAD INSERT,
         PROCEDURE my.t1_ud FOR UPLOAD DELETE,
         PROCEDURE my.t1_uu FOR UPLOAD UPDATE
      ),
      GROUPO.TABLE t2 AS my_t2 USING (
         PROCEDURE my.t2_ui FOR UPLOAD INSERT
      )
   );
```

The following example creates a download-only publication:

```
CREATE PUBLICATION p1 FOR DOWNLOAD ONLY (
   GROUPO.TABLE t1
);
```

Related Information

Publications
Publications and Articles
Scripted Upload
Download-only Publications
Publications for Scripted Upload
Publishing Only Some Rows in a Table (SQL Central)
Creating Publications (SQL Central)
Publishing Only Some Columns in a Table (SQL Central)
Publishing Only Some Rows Using the SUBSCRIBE BY Clause (SQL Central)
Publishing Only Some Rows Using a WHERE Clause (SQL Central)
Replicating the Primary Key Pool (SQL)
ALTER PUBLICATION Statement [MobiLink] [SQL Remote] [page 706]
DROP PUBLICATION Statement [MobiLink] [SQL Remote] [page 1111]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 997]
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 737]
SYSSYNC System View [page 1948]
Overlap Partitions

1.4.4.81 CREATE REMOTE [MESSAGE] Statement [SQL Remote]

Identifies a message link and return address for outgoing messages from a database.

Syntax

```
CREATE REMOTE [ MESSAGE ]
TYPE message-system
[ ADDRESS address-string ]
```

```
message-system : FILE | FTP | HTTP | SMTP
```

```
address-string : string
```

Parameters

message-system

One of the message systems supported by SQL Remote. It must be one of the following values: FILE, FTP, HTTP, or SMTP.

address-string

A string containing a valid address for the specified message system.

Remarks

The Message Agent sends outgoing messages from a database using one of the supported message links. Return messages for users employing the specified link are sent to the specified address as long as the remote database is created by the Extraction utility. The Message Agent starts links only if it has remote users for those links.

The address is the publisher's address under the specified message system. If it is an email system, the address string must be a valid email address. If it is a file-sharing system, the address string is a subdirectory of the directory set in the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

To remove the address, execute a CREATE REMOTE MESSAGE statement without an ADDRESS clause.

The dbinit utility creates message types automatically, without an address. Unlike other CREATE statements, the CREATE REMOTE MESSAGE statement does not give an error if the type exists; instead it alters the type.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

When remote databases are extracted using the extraction utility, the following statement sets all recipients of file message-system messages to send messages back to the `company` subdirectory.

The statement also instructs dbremote to look in the `company` subdirectory for incoming messages.

```
CREATE REMOTE MESSAGE TYPE FILE ADDRESS 'company';
```

The following statement sets the publisher's address for the SMTP message link to user@sample.com.

```
CREATE REMOTE MESSAGE TYPE SMTP ADDRESS 'user@sample.com';
```

Related Information

SQL Remote Message Systems
The FILE Message System
The FTP Message System
The SMTP Message System
Creating a Message Type (SQL Central)
GRANT PUBLISH Statement [SQL Remote] [page 1206]
GRANT REMOTE Statement [SQL Remote] [page 1208]
GRANT CONSOLIDATE Statement [SQL Remote] [page 1199]
DROP REMOTE [MESSAGE] Statement [SQL Remote] [page 1114]
ALTER REMOTE [MESSAGE] Statement [SQL Remote] [page 708]

1.4.4.82 CREATE ROLE Statement

Creates or replaces a role, creates a user-extended role, or modifies administrators for a role.

Syntax

```
CREATE [ OR REPLACE ] ROLE { role-name | FOR USER userid }
[ WITH ADMIN [ ONLY ] administrator-userid [,...] ]
```

Parameters

OR REPLACE clause

Use this clause to create the role if it does not already exist or replace its administrators if it does exist.

role-name

Use this parameter to specify the name of the role. This name must be unique across all users and roles in the database.

FOR USER userid clause

Use this clause to convert the specified user into a user-extended role that can be assigned to others. The user must not already be extended as another role.

WITH ADMIN and WITH ADMIN ONLY administrator-userid clause

Optionally specify administrators for the role. WITH ADMIN means that `administrator-userid` can exercise the role and administer it. WITH ADMIN ONLY means that `administrator-userid` can only administer the role. If no clause is specified, then any user with the MANAGE ROLES system privilege can administer the role.

The min_role_admins database option controls the minimum number of administrators required for each role. If you do not specify enough administrators when creating the role, the statement returns an error.

Remarks

The name of the new role must not begin and end with 'SYS_' and '_ROLE', respectively. For example SYS_MyBackup_ROLE is not an acceptable name for a user-defined role, whereas MyBackup_ROLE and SYS_MyBackup are acceptable.

If an ADMIN clause is specified, only the specified users can administer the roles. If no ADMIN clause is specified, then by default the role is granted to the MANAGE ROLES system privilege, with administrative rights only. This means that global administrators can administer the role.

To create a user-extended role (that is, to extend a user to be a role), use the `CREATE ROLE FOR USER userid` syntax.

Use the GRANT statements to grant system privileges to the role.

Privileges

You must have the MANAGE ROLES system privilege to create a new role.

If the OR REPLACE clause is specified and the role already exists, you must also have administrative rights over the role.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates the Sales role. Any user with the MANAGE ROLES system privilege can administer the role.

```
CREATE ROLE Sales;
```

The following statement extends user JaneSmith to become a role that can be assigned to others.

```
CREATE ROLE FOR USER JaneSmith;
```

The following statement creates the role Finance with MaryJones and JeffTurkott as role administrators with administrative rights (only) for the role.

```
CREATE ROLE Finance
WITH ADMIN ONLY MaryJones, JeffTurkott;
```

The following example replaces the existing Finance role created in the previous example, replacing MaryJones and JeffTurkott with EllenChong and DaveLexx as role administrators, this time with exercise rights to the role.

```
CREATE OR REPLACE ROLE Finance
WITH ADMIN EllenChong, DaveLexx;
```

Related Information

Role Administrators
User-extended Roles
Granting a Role (SQL Central)
min_role_admins Option
DROP ROLE Statement [page 1115]
ALTER ROLE Statement [page 710]

1.4.4.83 CREATE SCHEMA Statement

Creates a collection of tables and views for a database user.

Syntax

```
CREATE SCHEMA
 AUTHORIZATION userid
[  create-table-statement
   | create-view-statement
   | grant-statement
]  ... ;
```

Remarks

The CREATE SCHEMA statement creates a schema. A schema is a collection of tables and views along with their associated privileges.

The `userid` must be the user ID of the current connection. You cannot create a schema for another user.

If any statement contained in the CREATE SCHEMA statement fails, the entire CREATE SCHEMA statement is rolled back.

The CREATE SCHEMA statement is a way of collecting together individual CREATE and GRANT statements into one operation. There is no SCHEMA database object created in the database, and to drop the objects you must use individual DROP TABLE or DROP VIEW statements. To revoke privileges, you must use a REVOKE statement for each privilege granted.

The individual CREATE or GRANT statements are not separated by statement delimiters. The statement delimiter marks the end of the CREATE SCHEMA statement itself.

The individual CREATE or GRANT statements must be ordered such that the objects are created before privileges are granted on them.

Although you can create more than one schema for a user, doing so is not recommended.

Privileges

The system privileges required depend on the operation specified in the CREATE SCHEMA statement you define. For information about the required system privileges, see the System privileges sections for applicable statements (CREATE TABLE, CREATE VIEW, and GRANT).

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Core Feature. The ability to create multiple schemas for a single user is SQL optional Language Feature F171. The software does not support the use of REVOKE statements within the CREATE SCHEMA statement, and does not allow their use within Transact-SQL batches or procedures.

Transact-SQL

Supported by Adaptive Server Enterprise, which supports GRANT and REVOKE statements within the CREATE SCHEMA statement.

Example

The following CREATE SCHEMA statement creates a schema consisting of two tables. The statement must be executed by the user ID sample_user, who must have the CREATE TABLE system privilege. If the statement creating table t2 fails, neither table is created.

```
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY )
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

The statement delimiter in the following CREATE SCHEMA statement is placed after the first CREATE TABLE statement. As the statement delimiter marks the end of the CREATE SCHEMA statement, the example is interpreted as a two statement batch by the database server. If the statement creating table t2 fails, the table t1 is still created.

```
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY );
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

Related Information

CREATE TABLE Statement [page 1002]
CREATE VIEW Statement [page 1051]
GRANT Statement [page 1193]

1.4.4.84 CREATE SEMAPHORE Statement

Creates or replaces a semaphore and establishes the initial value for its counter. A semaphore is a locking mechanism that uses a counter to communicate and control the availability of a resource such as an external library or procedure.

Syntax

```
CREATE [ OR REPLACE | TEMPORARY ] SEMAPHORE [ IF NOT EXISTS ]
[ owner.]semaphore-name
[ START WITH initial-count ]
```

Parameters

owner

The owner of the semaphore. `owner` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

semaphore-name

The name of the semaphore. Specify a valid identifier in the CHAR database collation. `semaphore-name` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

OR REPLACE clause

Use this clause to overwrite (update) the definition of a permanent semaphore of the same name, if one exists.

If the OR REPLACE clause is specified, and a semaphore with this name is in use at the time, then the statement returns an error.

You cannot use this clause with the TEMPORARY or IF NOT EXISTS clauses.

TEMPORARY clause

Use this clause to create a temporary semaphore.

Do not use this clause with the OR REPLACE clause.

IF NOT EXISTS clause

Use this clause to create a semaphore only if it doesn't already exist. If a semaphore exists with the same name and same lifespan (permanent or temporary), then nothing happens and no error is returned.

You cannot use this clause with the OR REPLACE clause.

START WITH clause

Use this clause to specify the initial value for the semaphore counter. If this clause is not specified, then `initial-count` is set to 0.

`initial-count` can be specified using a variable (for example, **`START WITH @initial-count`**).

If you set `initial-count` to NULL, or if it is set to a variable and the variable value is NULL, the behavior is equivalent to not specifying the clause.

Remarks

The CREATE SEMAPHORE statement creates a semaphore and establishes a counter for it. Each time a NOTIFY SEMAPHORE statement is executed, the counter for the associated semaphore is incremented. Each time a WAITFOR SEMAPHORE statement is executed, and assuming the current count is a positive integer, the counter for the associated semaphore is decremented.

Permanent and temporary mutexes and semaphores share the same namespace, therefore you cannot create two of these objects with the same name. Use of the OR REPLACE and IF NOT EXISTS clause can inadvertently cause an error related to naming. For example, if you have a permanent mutex, and you try to create a temporary semaphore with the same name, an error is returned even if you specify IF NOT EXISTS. Similarly, if you have a temporary semaphore, and you try to replace it with a permanent semaphore with the same name by specifying OR REPLACE, an error is returned because this is equivalent to attempting to create a second object with the same name.

Permanent semaphore definitions persist across database restarts. However, their count returns to `initial-count` after a restart.

A temporary semaphore persists until the connection that created it is terminated, or until an explicit DROP operation is performed. If another connection is waiting for a temporary semaphore and the connection that created the temporary semaphore is terminated, then an error is returned to the waiting connection.

When replacing (OR REPLACE clause) a permanent semaphore, the old semaphore is deleted, and all connections waiting for the semaphore are notified.

If the OR REPLACE clause is specified, and a permanent semaphore with that name exists and connections are blocked waiting for the semaphore, the semaphore is still replaced. In this case, the waiting connections are unblocked and an error is returned to them indicating that the semaphore has been dropped. There is one exception however. If the replacement semaphore definition has identical settings, there is no impact to waiting connections.

Privileges

Requires the CREATE ANY MUTEX SEMAPHORE system privilege.

Side Effects

Automatic commit, but only for permanent semaphores.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a semaphore called license_counter and sets its counter to 3:

```
CREATE SEMAPHORE license_counter START WITH 3;
```

Related Information

Mutexes and Semaphores
DROP SEMAPHORE Statement [page 1118]
NOTIFY SEMAPHORE Statement [page 1282]
WAITFOR SEMAPHORE Statement [page 1483]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.85 CREATE SEQUENCE Statement

Creates a sequence that can be used to generate primary key values that are unique across multiple tables, and for generating default values for a table.

Syntax

```
CREATE [ OR REPLACE ] SEQUENCE [ owner .] sequence-name
[ INCREMENT BY signed-integer ]
[ START WITH signed-integer ]
[ MINVALUE signed-integer | NO MINVALUE ]
[ MAXVALUE signed-integer | NO MAXVALUE ]
[ CACHE integer | NO CACHE ]
[ CYCLE | NO CYCLE ]
```

Parameters

OR REPLACE clause

Specifying OR REPLACE creates a new sequence, or replaces an existing sequence with the same name. If you do not use the OR REPLACE clause, an error is returned if you specify the name of a sequence that already exists for the current user.

INCREMENT BY clause

Defines the amount the next sequence value is incremented from the last value assigned. The default is 1. Specify a negative value to generate a descending sequence. An error is returned if the INCREMENT BY value is 0.

START WITH clause

Defines the starting sequence value. If you do not specify a value for the START WITH clause, MINVALUE is used for ascending sequences and MAXVALUE is used for descending sequences. An error is returned if the START WITH value is beyond the range specified by MINVALUE or MAXVALUE.

MINVALUE clause

Defines the smallest value generated by the sequence. The default is 1. An error is returned if MINVALUE is greater than (2^63-1) or less than -(2^63-1). An error is also returned if MINVALUE is greater than MAXVALUE.

MAXVALUE clause

Defines the largest value generated by the sequence. The default is 2^63-1. An error is returned if MAXVALUE is greater than 2^63-1 or less than -(2^63-1).

CACHE clause

Specifies the number of preallocated sequence values that are kept in memory for faster access. When the cache is exhausted, the sequence cache is repopulated and a corresponding entry is written to the transaction log. At checkpoint time, the current value of the cache is forwarded to the ISYSSEQUENCE system table. The default is 100.

CYCLE clause

Specifies whether values should continue to be generated after the maximum or minimum value is reached.

The default is NO CYCLE, which returns an error once the maximum or minimum value is reached.

Remarks

A sequence is a database object that allows the automatic generation of numeric values. A sequence is not bound to a specific or unique table column.

Sequences can generate values in one of the following ways:

- Increment or decrement monotonically without bound
- Increment or decrement monotonically to a user-defined limit and stop
- Increment or decrement monotonically to a user-defined limit and cycle back to the beginning and start again

You control the behavior when the sequence runs out of values using the CYCLE clause.

If a sequence is increasing and it exceeds the MAXVALUE, MINVALUE is used as the next sequence value if CYCLE is specified. If a sequence is decreasing and it falls below MINVALUE, MAXVALUE is used as the next sequence value if CYCLE is specified. If CYCLE is not specified, an error is returned.

Sequence values cannot be used with views or materialized view definitions.

To return the next value in the sequence, use `[owner.]sequence-name.NEXTVAL`. The sequence is shared by all connections, so each connection will get a unique next value.

To return the most recently supplied sequence value for the current connection, use `[owner.]sequence-name.CURRVAL`.

Privileges

Requires one of:

- CREATE ANY SEQUENCE or system privilege
- CREATE ANY OBJECT system privilege

To replace an existing sequence, requires one of:

- CREATE ANY SEQUENCE and DROP ANY SEQUENCE system privileges
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges
- ALTER ANY OBJECT or ALTER ANY SEQUENCE system privileges

Side Effects

None

Standards

ANSI/ISO SQL Standard

Sequences comprise SQL Language Feature T176. The software does not allow optional specification of the sequence data type. This behavior can be achieved with a CAST when using the sequence.

In addition, the following are not in the standard:

- CACHE clause
- OR REPLACE syntax
- CURRVAL expression
- Use of sequences in DEFAULT expressions

Example

The following example creates a sequence named Test that starts at 4, increments by 2, does not cycle, and caches 15 values at a time:

```
CREATE SEQUENCE Test
START WITH 4
INCREMENT BY 2
NO MAXVALUE
NO CYCLE
CACHE 15;
```

Related Information

Use of a Sequence to Generate Unique Values
ALTER SEQUENCE Statement [page 712]
DROP SEQUENCE Statement [page 1119]

1.4.4.86 CREATE SERVER Statement

Creates a remote server or a directory access server.

Syntax

Create a remote server

```
CREATE [ REMOTE ] SERVER server-name
CLASS server-class-string | variable
USING connection-info-string | variable
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ DEFAULT LOGIN string | variable [ IDENTIFIED BY string | variable ] ]
```

```
server-class-string :
{ 'ADSODBC' | 'ADS_ODBC'
| 'ASEODBC' | 'ASE_ODBC'
| 'DB2ODBC' | 'DB2_ODBC'
| 'HANAODBC' | 'HANA_ODBC'
| 'IQODBC' | 'IQ_ODBC'
| 'MIRROR'
| 'MSACCESSODBC' | 'MSACCESS_ODBC'
| 'MSSODBC' | 'MSS_ODBC'
| 'MYSQLODBC' | 'MYSQL_ODBC'
| 'ODBC'
| 'ORAODBC' | 'ORA_ODBC'
| 'SAODBC' | 'SA_ODBC'
| 'ULODBC' | 'UL_ODBC' }
```

```
connection-info-string :
  { 'data-source-name' | 'connection-string' }
```

Create a remote server (SAP HANA syntax)

```
CREATE REMOTE SOURCE remote-source-name
   ADAPTER adapter-name | variable
   CONFIGURATION connection-info-string | variable
   [ READ ONLY [ ON | OFF | VALUE variable ] ]
   [ WITH CREDENTIAL TYPE { 'PASSWORD' | variable } USING { 'USER=remote-
user;PASSWORD=remote-password' | variable ]
```

```
connection-info-string :
  { 'data-source-name' | 'connection-string' }
```

Create a directory access server

```
CREATE SERVER server-name
CLASS 'DIRECTORY'
USING using-string | variable
```

```
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ ALLOW { 'ALL' | 'SPECIFIC' | variable } USERS ]
```

```
using-string :
  'ROOT= path [ ;SUBDIRS= n ] [ ;CREATEDIRS= { YES | NO } ] [ ;DELIMITER=
{ / | \ } ]'
```

Parameters

CREATE [REMOTE] SERVER

The REMOTE keyword is optional and is provided for compatibility with other databases.

CLASS clause

Specifies the server class you want to use for a remote connection. Server classes contain detailed server capability information.

The DIRECTORY class is used to create a directory access server that accesses a directory on the local computer.

The server classes are:

SAODBC

for SQL Anywhere.

ULODBC

for UltraLite.

> **i** Note
>
> You cannot create a remote server for an UltraLite database running on macOS.

ADSODBC

for SAP Advantage Database Server.

ASEODBC

for SAP Adaptive Server Enterprise (version 10 and later).

DB2ODBC

for IBM DB2.

HANAODBC

for SAP HANA.

IQODBC

for SAP IQ.

MSACCESSODBC

for Microsoft Access.

MSSODBC

for Microsoft SQL Server.

MYSQLODBC

for Oracle MySQL.

ODBC

for all other ODBC data sources.

ORAODBC

for Oracle Database servers (version 8.0 and later).

> **i** Note
>
> When using remote data access, if you use an ODBC driver that does not support Unicode, then character set conversion is not performed on data coming from that ODBC driver.

READ ONLY clause (remote server)

Specifies that the remote server is accessed in read-only mode. If the clause is not specified, or if READ ONLY OFF is specified, then the remote server is not accessed in read-only mode. If READ ONLY or READ ONLY ON is specified, then the remote server is accessed in read-only mode.

READ ONLY clause (directory access server)

Specifies whether the files accessed by the directory are read-only and cannot be modified. By default, READ ONLY is set to NO.

ALLOW USERS clause (directory access servers)

Specify ALLOW 'SPECIFIC' USERS to restrict access to the directory access server to users with an external login. You must explicitly create an external login to the directory access server for each user that needs access. This clause is the default.

Specify the ALLOW 'ALL' USERS clause if you are not concerned about who has access to the directory access server, or you want everyone in your database to have access. This clause creates a default external login for the directory access server that is available to all users.

USING clause (remote servers)

When you create a remote server, the USING clause supplies a connection string for the database server. The appropriate connection string depends on the driver being used, which in turn depends on the value specified.

The USING clause is an ODBC connection string that can include `'DSN=data-source-name'` to specify an ODBC data source name and/or `'DRIVER=driver-name'` to specify a driver binary on UNIX/Linux or a driver name on Microsoft Windows.

For SQL Anywhere remote servers (SAODBC server classes), the `connection-info-string` parameter can be any valid connection string. Use any supported connection parameter. For example, if you have connection problems, then include a LOG connection parameter to troubleshoot the connection attempt.

The string in the USING clause can also contain local or global variable names enclosed in braces ({`variable-name`}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains `'DSN={@mydsn}'` indicates that `@mydsn` is a SQL variable and that the current contents of the `@mydsn` variable should be substituted when a connection is made to the remote data access server.

USING clause (directory access servers)

Specify the server connection information.

When you create a directory access server, the USING clause specifies the following values for the local directory:

ROOT clause

Specifies the path, relative to the database server, that is the root of the directory access class. When you create a proxy table using the directory access server name, the proxy table is relative to this root path.

SUBDIRS clause

Specifies a number between 0 and 10 that represents the number of levels of directories within the root that the database server can access. If SUBDIRS is omitted or set to 0, then only the files in the root directory are accessible via the directory access server. You can create proxy tables to any of the directories or subdirectories available via the directory access server.

CREATEDIRS clause

Specifies whether directories can be created using the directory access server. The default is NO.

DELIMITER clause

Specifies whether paths are delimited by a slash (/) or backslash (\) character. By default, the native path delimiter is used.

The string in the USING clause can also contain local or global variable names enclosed in braces ({`variable-name`}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains `'ROOT={@mypath}'` indicates that `@mypath` is a SQL variable and that the current contents of the `@mypath` variable should be substituted when a connection to the directory access server is established.

DEFAULT LOGIN clause (remote server)

Specifies a default user ID and (optionally) password for an account on the remote server that is to be used as the default login. Values for the DEFAULT LOGIN clause are restricted to 128 bytes.

IDENTIFIED BY clause

The IDENTIFIED BY clause specifies the remote password for the remote user. This value can be either a string or a variable. The remote user and remote password combination must be valid on the remote server.

If you omit the IDENTIFIED BY clause, the password is sent to the remote server as NULL. However, if you specify IDENTIFIED BY "" (an empty string), then the password sent is the empty string.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

CREATE REMOTE SOURCE (SAP HANA syntax)

This syntax, including the parameters and their values, is semantically equivalent to the syntax for creating a remote server (CREATE [REMOTE]) syntax and is provided for compatibility with SAP HANA servers. There is a one-to-one clause match between the two syntaxes as follows:

ADAPTER adapter-name

See the CLASS clause description for the CREATE [REMOTE] SERVER syntax.

CONFIGURATION connection-info-string

See the USING clause description for the CREATE [REMOTE] SERVER syntax.

READ ONLY [ON | OFF] clause

See the READ ONLY clause description for the CREATE [REMOTE] SERVER syntax.

WITH CREDENTIAL TYPE clause

See the DEFAULT LOGIN clause description for the CREATE [REMOTE] SERVER syntax.

Remarks (remote server)

Use the CREATE SERVER statement to create a remote server to access to data in other data sources. Once you create a remote server, you must create local proxy tables to map to the remote tables. Database users use proxy tables to access the contents of the remote tables. Create an external login for each database user that needs to communicate with the remote server.

Connections to a remote server are first attempted using the current executing user's external login. If this user does not have an external login, then the connection is attempted using the DEFAULT LOGIN credentials. If the remote server was created without a DEFAULT LOGIN, and no external login has been defined for the user, then the connection is attempted with the current executing user ID and password.

When running procedures that involve connections to a remote server, you can use the extern_login_credentials option to specify whether the remote data access connections are performed using the external login credentials of the logged in user or the effective user.

On UNIX or Linux, the database server runs as a specific user, so file permissions are based on the privileges granted to the database server user.

When accessing data from a remote server, if you use an ODBC driver that does not support Unicode, then character set conversion is not performed on data coming from that ODBC drive

When you define a remote server, an entry is added to the ISYSSERVER system table for the remote server. View the list of remote servers by querying the SYSSERVER system view.

> **i** Note
>
> For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Remarks (directory access server)

Use the CREATE SERVER statement to create a directory access server that accesses the local directory structure on the computer where the database server is running. Create an external login for each database user that needs to use the directory access server. Once you create a directory access server, you must create a proxy table for it. Database users use proxy tables to access the contents of a directory on the database server's local file system.

View the list of directory access servers by querying the SYSSERVER system view.

Privileges

You must have the SERVER OPERATOR system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a SQL Anywhere remote server named RemoteSA using the SQL Anywhere ODBC driver:

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere 17;DSN=RemoteDS';
```

The following example directly loads the SQL Anywhere ODBC driver without using the ODBC driver manager:

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere Native;DSN=RemoteDS';
```

The following example uses a variable reference to create a dynamic remote data access server. You need the MANAGE ANY USER and CREATE TABLE system privileges to run this example.

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere 17;DSN={dsn_string};Server=saremote;UID=DBA;PWD=sql';
CREATE VARIABLE dsn_string LONG VARCHAR;
SET dsn_string = 'Test17';
CREATE EXTERNLOGIN DBA TO RemoteSA;
CREATE EXISTING TABLE test_employees
AT 'RemoteSA...Employees';
SELECT * FROM test_employees;
DROP REMOTE CONNECTION TO RemoteSA CLOSE ALL;
```

The following example creates an Adaptive Server Enterprise (ASE) remote server named ase_prod using the ASE ODBC driver:

```
CREATE SERVER ase_prod
CLASS 'ASEODBC'
USING 'DSN=remoteASE';
```

The following example creates a remote server for the Oracle server named oracle723. Its ODBC data source name is oracle723.

```
CREATE SERVER oracle723
CLASS 'ORAODBC'
USING 'oracle723';
```

The following example creates a directory access server that only sees files within the directory `c:\temp`:

```
CREATE SERVER diskserver0
CLASS 'DIRECTORY'
USING 'ROOT=c:\\temp';
CREATE EXTERNLOGIN DBA TO diskserver0;
CREATE EXISTING TABLE diskdir0 AT 'diskserver0;;;.';
-- Get a list of those files.
SELECT privileges, file_name, size FROM diskdir0;
```

The following example creates a dynamic directory access server that is used to explore two different directories:

```
CREATE SERVER diskserver9
CLASS 'DIRECTORY'
USING '{dir_options}';
CREATE EXTERNLOGIN DBA TO diskserver9;
CREATE EXISTING TABLE diskdir9 AT 'diskserver9;;;.';
CREATE VARIABLE dir_options VARCHAR(256);
SET dir_options = 'ROOT=c:\\temp;SUBDIRS=9;DELIMITER=/';
SELECT * FROM diskdir9;
DROP REMOTE CONNECTION TO diskserver9 CLOSE ALL;
SET dir_options = 'ROOT=c:\\ProgramData;SUBDIRS=9;DELIMITER=/';
SELECT * FROM diskdir9;
```

When you create a remote server, to bypass the ODBC driver manager, use the syntax below, followed by the remainder of the `connection-info-string`:

```
CREATE SERVER remote-server
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere Native;DSN=my-dsn;UID=my-username;PWD=my-pwd';
```

This syntax allows remote data access to load the SQL Anywhere ODBC driver directly and is supported by Microsoft Windows and UNIX/Linux. Loading the SQL Anywhere ODBC driver directly ensures that the ODBC driver for the current server version is used. Also, if the SQL Anywhere ODBC driver is only used for remote data access, then it does not need to be registered.

On UNIX platforms you can also reference the SQL Anywhere ODBC driver. The syntax is as follows:

```
USING 'DRIVER=SQL Anywhere 17;DSN=my-dsn'
```

> **Note**
>
> If the application also makes use of non-SQL Anywhere remote servers, or if there are SQL Anywhere remote servers defined without using `'DRIVER=SQL Anywhere Native'`, then remote data access still uses a driver manager for the other remote servers.

Related Information

Remote Data Access
Remote Servers and Remote Table Mappings
Directory Access Servers
Server Classes for Remote Data Access
Alphabetical List of Connection Parameters
Creating Directory Access Servers (SQL)
extern_login_credentials Option
ALTER SERVER Statement [page 714]
CREATE EXTERNLOGIN Statement [page 858]
CREATE EXISTING TABLE Statement [page 854]
DROP SERVER Statement [page 1120]
DROP REMOTE CONNECTION Statement [page 1112]
SYSSERVER System View [page 1943]

1.4.4.87 CREATE SERVICE Statement [HTTP Web Service]

Creates a new HTTP web service.

> **Syntax**

```
CREATE [ OR REPLACE ] SERVICE service-name
TYPE { 'RAW' | 'HTML' | 'JSON' | 'XML' }
[ URL [PATH] { ON | OFF | ELEMENTS } ]
[ common-attributes ]
[ AS { statement | NULL } ]
```

```
common-attributes :
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

```
method :
DEFAULT
| POST
| GET
| HEAD
| OPTIONS
```

```
| PUT
| DELETE
| NONE
| *
```

Parameters

service-name

Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

You can name your service root, but this name has a special function.

TYPE clause

Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

'RAW'

The result set is sent to the client without any formatting. Utilization of this service requires that all content markup is explicitly provided. Complex dynamic content containing current content with markup, JavaScript and images can be generated on demand. The media type may be specified by setting the Content-Type response header using the sa_set_http_header procedure. Setting the Content-Type header to 'text/html' is good practice when generating HTML markup to ensure that all browsers display the markup as HTML and not text/plain.

'HTML'

The result set is returned as an HTML representation of a table or view.

'JSON'

The result set is returned in JavaScript Object Notation (JSON). A JSON service does not automatically process JSON input. It only presents data (in the response) in JSON format. JSON accepts POST/PUT methods where application/x-www-form-urlencoded is supported. If for a POST/PUT method, `Content-Type: application/json` is specified, then the application may use http_variable('body') to retrieve the JSON (request) content. The database server does not parse the JSON input automatically. It is up to the application to parse it.

'XML'

The result set is returned as XML. If the result set is already XML, no additional formatting is applied. Otherwise, it is automatically formatted as XML. As an alternative approach, a RAW service could return a select using the FOR XML RAW clause having set a valid Content-Type such as text/xml using sa_set_http_header procedure.

URL clause

Determines whether URL paths are accepted and, if so, how they are processed. Specifying URL PATH has the same effect as URL.

OFF

Indicates that the service name in a URL request must not be followed by a path. OFF is the default setting. For example, the following form is disallowed due to the path elements `/aaa/bbb/ccc`.

```
http://host-name/service-name/aaa/bbb/ccc
```

Suppose that `CREATE SERVICE echo URL PATH OFF` was specified when creating the web service. A URL similar to `http://localhost/echo?id=1` produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_HEADER('@HttpQueryString')	id=1

ON

Indicates that the service name in a URL request can be followed by a path. The path value is returned by querying a dedicated HTTP variable named URL. A service can be defined to explicitly provide the URL parameter or it may be retrieved using the HTTP_VARIABLE function. For example, the following form is allowed:

```
http://host-name/service-name/aaa/bbb/ccc
```

Suppose that `CREATE SERVICE echo URL PATH ON` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_VARIABLE('URL')	one/two
HTTP_HEADER('@HttpQueryString')	id=1

ELEMENTS

Indicates that the service name in a URL request may be followed by a path. The path is obtained in segments by specifying a single parameter keyword **`URL1`**, **`URL2`**, and so on. Each parameter may be retrieved using the HTTP_VARIABLE or NEXT_HTTP_VARIABLE functions. These iterator functions can be used in applications where a variable number of path elements can be provided. For example, the following form is allowed:

```
http://host-name/service-name/aaa/bbb/ccc
```

Suppose that `CREATE SERVICE echo URL PATH ELEMENTS` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_VARIABLE('URL1')	one
HTTP_VARIABLE('URL2')	two
HTTP_VARIABLE('URL3')	NULL

Function call	Result
HTTP_HEADER('@HttpQueryString')	id=1

Up to 10 elements can be obtained. A NULL value is returned if the corresponding element is not supplied. In the above example, `HTTP_VARIABLE('URL3')` returns NULL because no corresponding element was supplied.

AUTHORIZATION clause

Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for all services and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) that obfuscates the user and password information using base-64 encoding. Use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses

Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a `404 Not Found` HTTP status.

METHODS clause

Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, OPTIONS, PUT, DELETE, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods which are default request types for the RAW, HTML, and XML service types. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	all	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	RAW, HTML, JSON, XML	Enabled by default.
GET	RAW, HTML, JSON, XML	Enabled by default.
HEAD	RAW, HTML, JSON, XML	Enabled by default.
OPTIONS	RAW, HTML, JSON, XML	Not enabled by default.
PUT	RAW, HTML, JSON, XML	Not enabled by default.
DELETE	RAW, HTML, JSON, XML	Not enabled by default.
NONE	all	Use NONE to disable access to a service.

Method value	Applies to service	Description
*	RAW, HTML, JSON, XML	Same as specifying `'POST,GET,HEAD'`.

For example, you can use either of the following clauses to specify that a service supports all HTTP method types:

```
METHODS '*,OPTIONS,PUT,DELETE'
METHODS 'POST,GET,HEAD,OPTIONS,PUT,DELETE'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause

Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause

Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a `401 Authorization Required` HTTP response status. Based on this response, the web browser prompts for a user ID and password.

> ⚠ Caution
>
> It is strongly recommended that you specify a USER clause when authorization is enabled (default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement

Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

An HTTP request to a non-DISH service with no `statement` specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use `statement` to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following `statement` demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

Service definitions are stored within the ISYSWEBSERVICE table and can be examined from the SYSWEBSERVICE system view.

If a USER clause is specified, the service is dropped if `user-name` is dropped.

Privileges

You must have the MANAGE ANY WEB SERVICE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

CREATE SERVICE is supported by Adaptive Server Enterprise, for XML and RAW types only.

Example

The following example demonstrates how to create a JSON service.

Start a database server with the -xs (http or https) option and then execute the following SQL statements to set up the service:

```
CREATE PROCEDURE ListEmployees()
RESULT (
 EmployeeID            integer,
 Surname               person_name_t,
 GivenName             person_name_t,
 StartDate             date,
 TerminationDate       date )
BEGIN
    SELECT EmployeeID, Surname, GivenName, StartDate, TerminationDate
    FROM GROUPO.Employees
END;
CREATE SERVICE "jsonEmployeeList"
    TYPE 'JSON'
    AUTHORIZATION OFF
```

```
    SECURE OFF
    USER DBA
    AS CALL ListEmployees();
```

The JSON service provides data for easy consumption by an AJAX call back.

Execute the following SQL statement to create an HTML service that provides the service in a readable form:

```
CREATE SERVICE "EmployeeList"
    TYPE 'HTML'
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA
    AS CALL ListEmployees();
```

Use a web browser to access the service using a URL similar to `http://localhost/EmployeeList`.

Related Information

How to Access Client-supplied HTTP Variables and Headers
How to Access Client-supplied HTTP Variables and Headers
The Database Server as an HTTP Web Server
How to Create and Customize a Root Web Service
How to Develop Web Service Applications in an HTTP Web Server
How to Browse a SQL Anywhere HTTP Web Server
ALTER SERVICE Statement [HTTP Web Service] [page 718]
DROP SERVICE Statement [page 1122]
sp_parse_json System Procedure [page 1799]
SYSWEBSERVICE System View [page 1972]
sa_set_http_header System Procedure [page 1711]
Identifiers [page 6]
ROW Constructor [Composite] [page 534]
ARRAY Constructor [Composite] [page 243]

1.4.4.88 CREATE SERVICE Statement [SOAP Web Service]

Creates a new SOAP over HTTP or DISH service.

Syntax

SOAP over HTTP services

```
CREATE [ OR REPLACE ] SERVICE service-name
TYPE 'SOAP'
[ DATATYPE { ON | OFF | IN | OUT } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
[ common-attributes ]
AS statement
```

```
common-attributes :
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

```
method :
DEFAULT
| POST
| HEAD
| NONE
```

DISH services

```
CREATE SERVICE service-name
TYPE 'DISH'
[ GROUP { group-name | NULL } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
[ common-attributes ]
```

```
common-attributes:
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

```
method :
DEFAULT
| POST
| GET
| HEAD
| NONE
| *
```

Parameters

service-name

Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

Unlike other services, you cannot use a slash (/) anywhere in a DISH service name.

You can name your service root, but this name has a special function.

TYPE clause

Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

'SOAP'

The result set is returned as an XML payload known as a SOAP envelope. The format of the data may be further refined using by the FORMAT clause. A request to a SOAP service must be a valid SOAP request, not just a general HTTP request.

'DISH'

A DISH service (Determine SOAP Handler) is a SOAP endpoint that references any SOAP service within its GROUP context. It also exposes the interfaces to its SOAP services by generating a WSDL (Web Services Description Language) for consumption by SOAP client toolkits.

GROUP clause

A DISH service without a GROUP clause exposes all SOAP services defined within the database. By convention, the SOAP service name can be composed of a GROUP and a NAME element. The name is delimited from the group by the last slash character. For example, a SOAP service name defined as 'aaa/bbb/ccc' is 'ccc', and the group is 'aaa/bbb'. Delimiting a DISH service using this convention is invalid. Instead, a GROUP clause is applied to specify the group of SOAP services for which it is to be the SOAP endpoint.

> **i** Note
>
> Slashes are converted to underscores within the WSDL to produce valid XML. Use caution when using a DISH service that does not specify a GROUP clause such that it exposes all SOAP services that may contain slashes. Use caution when using groups with SOAP service names that contain underscores to avoid ambiguity.

DATATYPE clause

Applies to SOAP services only. When DATATYPE OFF is specified, SOAP input parameters and response data are defined as XMLSchema string types. In most cases, true data types are preferred because it negates the need for the SOAP client to cast the data prior to computation. Parameter data types are exposed in the schema section of the WSDL generated by the DISH service. Output data types are represented as XML schema type attributes for each column of data.

The following values are permitted for the DATATYPE clause:

ON

Generates data typing of input parameters and result set responses.

OFF

All input parameters and response data are typed as XMLSchema string (default).

IN

Generates true data types for input parameters only. Response data types are XMLSchema string.

OUT

Generates true data types for responses only. Input parameters are typed as XMLSchema string.

FORMAT clause

This clause specifies the output format when sending responses to SOAP client applications.

The SOAP service format is dictated by the associated DISH service format specification when it is not specified by the SOAP service. The default format is DNET.

SOAP requests should be directed to the DISH service (the SQL Anywhere SOAP endpoint) to leverage common formatting rules for a group of SOAP services (SOAP operations). A SOAP service FORMAT specification overrides that of a DISH service. The format specification of the DISH service is used when a

SOAP service does not define a FORMAT clause. If no FORMAT is provided by either service then the default is 'DNET'.

The following formats are supported:

'DNET'

The output is in a System.Data.DataSet compatible format for consumption by .NET client applications. (default)

'CONCRETE'

This output format is used to support client SOAP toolkits that are capable of generating interfaces representing arrays of row and column objects but are not able to consume the DNET format. Java and .NET clients can easily consume this output format.

The specific format is exposed within the WSDL as an explicit dataset object or a SimpleDataset. Both dataset representations describe a data structure representing an array of rows with each row containing an array of columns. An explicit dataset object has the advantage of representing the actual shape of the result set by providing parameter names and data types for each column in the row. In contrast, the SimpleDataset exposes rows containing an unbounded number of columns of any type.

FORMAT 'CONCRETE' EXPLICIT ON requires that the Service statement calls a stored procedure which defines a RESULT clause. Having met this condition, the SOAP service will expose an explicit dataset whose name begins with the service name appended with Dataset.

If the condition is not met, a SimpleDataset is used.

'XML'

The output is generated in an XMLSchema string format. The response is an XML document that requires further processing by the SOAP client to extract column data. This format is suitable for SOAP clients that cannot generate intermediate interface objects that represent arrays of rows and columns.

NULL

A NULL type causes the SOAP or DISH service to use the default behavior. The format type of an existing service is overwritten when using the NULL type in an ALTER SERVICE statement.

AUTHORIZATION clause

Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for SOAP services, and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) which obfuscates the user and password information using base-64 encoding. It is recommended that you use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses

Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a `404 Not Found` HTTP status.

METHODS clause

Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods. The default method types for SOAP services are POST and HEAD. The default method types for DISH services are GET, POST, and HEAD. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	both	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	both	Enabled by default for SOAP.
GET	DISH only	Enabled by default for DISH.
HEAD	both	Enabled by default for SOAP and DISH.
NONE	both	Use NONE to disable access to a service. When applied to a SOAP service, the service cannot be directly accessed by a SOAP request. This enforce exclusive access to a SOAP operation through a DISH service SOAP endpoint. It is recommended that you specify **`METHODS 'NONE'`** for each SOAP service.
*	DISH only	Same as specifying **`'POST,GET,HEAD'`**.

For example, you can use the following clause to specify that a service supports all SOAP over HTTP method types:

```
METHODS 'POST,HEAD'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause

Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause

Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be

specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a `401 Authorization Required` HTTP response status. Based on this response, the web browser prompts for a user ID and password.

> ⚠ Caution
>
> It is strongly recommended that you specify a USER clause when authorization is enabled (the default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement

Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

A DISH service is the only service that must either define a null statement, or not define a statement. A SOAP service must define a statement. Any other SERVICE can have a NULL statement, but only if it is configured with AUTHORIZATION ON.

An HTTP request to a non-DISH service with no `statement` specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use `statement` to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following `statement` demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

Service definitions are stored within the ISYSWEBSERVICE table and can be examined from the SYSWEBSERVICE view.

Privileges

You must have the MANAGE ANY WEB SERVICE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

CREATE SERVICE is supported by Adaptive Server Enterprise for SOAP types only.

Related Information

How to Access Client-supplied HTTP Variables and Headers
SOAP Data Types
The Database Server as an HTTP Web Server
How to Create and Customize a Root Web Service
Tutorial: Using a Database Server to Access a SOAP/DISH Service
ALTER SERVICE Statement [SOAP Web Service] [page 724]
DROP SERVICE Statement [page 1122]
SYSWEBSERVICE System View [page 1972]

1.4.4.89 CREATE SPATIAL REFERENCE SYSTEM Statement

Creates or replaces a spatial reference system.

Syntax

```
{ CREATE [ OR REPLACE ] SPATIAL REFERENCE SYSTEM
| CREATE SPATIAL REFERENCE SYSTEM IF NOT EXISTS }
srs-name
[ srs-attribute [ srs-attribute ... ]]
```

```
srs-name : string
```

```
srs-attribute :
IDENTIFIED BY srs-id
| DEFINITION { definition-string | NULL }
| ORGANIZATION { organization-name IDENTIFIED BY organization-srs-id | NULL }
| TRANSFORM DEFINITION { transform-definition-string | NULL }
| LINEAR UNIT OF MEASURE linear-unit-name
| ANGULAR UNIT OF MEASURE { angular-unit-name | NULL }
| TYPE { ROUND EARTH | PLANAR }
| COORDINATE coordinate-name { UNBOUNDED | BETWEEN low-number AND high-
number }
| ELLIPSOID SEMI MAJOR AXIS semi-major-axis-length { SEMI MINOR AXIS semi-minor-
axis-length | INVERSE FLATTENING inverse-flattening-ratio }
| SNAP TO GRID { grid-size | DEFAULT }
| TOLERANCE { tolerance-distance | DEFAULT }
| AXIS ORDER axis-order
| POLYGON FORMAT polygon-format
```

```
| STORAGE FORMAT storage-format

srs-id : integer

semi-major-axis-length : number

semi-minor-axis-length : number

inverse-flattening-ratio : number

grid-size : DOUBLE : usually between 0 and 1

tolerance-distance : number

axis-order : { 'x/y/z/m' | 'long/lat/z/m' | 'lat/long/z/m' }

polygon-format : { 'CounterClockWise' | 'Clockwise' | 'EvenOdd' }

storage-format : { 'Internal' | 'Original' | 'Mixed' }
```

Parameters

OR REPLACE clause

Specifying OR REPLACE creates the spatial reference system if it does not already exist in the database, and replaces it if it does exist. An error is returned if you attempt to replace a spatial reference system while it is in use. An error is also returned if you attempt to replace a spatial reference system that already exists in the database without specifying the OR REPLACE clause.

CREATE SPATIAL REFERENCE IF NOT EXISTS

Specifying CREATE SPATIAL REFERENCE IF NOT EXISTS checks to see if a spatial reference system by that name already exists. If it does not exist, the database server creates the spatial reference system. If it does exist, no further action is performed and no error is returned.

IDENTIFIED BY clause

Use this clause to specify the SRID (`srs-id`) for the spatial reference system. If the spatial reference system is defined by an organization with an `organization-srs-id`, then `srs-id` should be set to that value.

If the IDENTIFIED BY clause is not specified, then the SRID defaults to the `organization-srs-id` defined by either the ORGANIZATION clause or the DEFINITION clause. If neither clause defines an `organization-srs-id` that could be used as a default SRID, an error is returned.

When the spatial reference system is based on a well known coordinate system, but has a different geodesic interpretation, set the `srs-id` value to be 1000000000 (one billion) plus the well known value. For example, the SRID for a planar interpretation of the geodetic spatial reference system WGS 84 (ID 4326) would be 1000004326.

With the exception of SRID 0, spatial reference systems provided by SQL Anywhere that are not based on well known systems are given a SRID of 2000000000 (two billion) and above. The range of SRID values from 2000000000 to 2147483647 is reserved by SQL Anywhere and you should not create SRIDs in this range.

To reduce the possibility of choosing a SRID that is reserved by a defining authority such as OGC or by other vendors, you should not choose a SRID in the range 0 - 32767 (reserved by EPSG), or in the range 2147483547 - 2147483647.

Also, since the SRID is stored as a signed 32-bit integer, the number cannot exceed 2^{31}-1 or 2147483647.

DEFINITION clause

Use this clause to set, or override, default coordinate system settings. If any attribute is set in a clause other than the DEFINITION clause, it takes the value specified in the other clause regardless of what is specified in the DEFINITION clause.

`definition-string` is a string in the Spatial Reference System Well Known Text syntax as defined by SQL/MM and OGC. For example, the following query returns the definition for WGS 84.

```
SELECT  ST_SpatialRefSys::ST_FormatWKT( definition )
   FROM ST_SPATIAL_REFERENCE_SYSTEMS
   WHERE srs_id=4326;
```

In Interactive SQL, if you double-click the value returned, an easier to read version of the value appears.

When the DEFINITION clause is specified, `definition-string` is parsed and used to choose default values for attributes. For example, `definition-string` may contain an AUTHORITY element that defines the `organization-name` and `organization-srs-id`.

Parameter values in `definition-string` are overridden by values explicitly set using the SQL statement clauses. For example, if the ORGANIZATION clause is specified, it overrides the value for ORGANIZATION in `definition-string`.

ORGANIZATION clause

Use this clause to specify information about the organization that created the spatial reference system that the new spatial reference system is based on. `organization-name` is the name of the organization that created it; `organization-srs-id` is the numeric identifier the organization uses to identify the spatial reference system.

TRANSFORM DEFINITION clause

Use this clause to specify a description of the transform to use for the spatial reference system. Currently, only the PROJ.4 transform is supported. For example, the `transform-definition-string` for WGS 84 is `'+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'`.

If you specify an unsupported transform definition, an error is returned.

The transform definition is used by the ST_Transform method when transforming data between spatial reference systems. Some transforms may still be possible even if there is no `transform-definition-string` defined.

LINEAR UNIT OF MEASURE clause

Use this clause to specify the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure that is defined in the ST_UNITS_OF_MEASURE consolidated view.

If this clause is not specified, and is not defined in the DEFINITION clause, the default is METRE.

To add predefined units of measure to the database, use the sa_install_feature system procedure.

To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

> **i** Note
>
> While both METRE and METER are accepted spellings, METRE is preferred as it conforms to the SQL/MM standard.

ANGULAR UNIT OF MEASURE clause

Use this clause to specify the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the ST_UNITS_OF_MEASURE consolidated view.

If this clause is not specified, and is not defined in the DEFINITION clause, the default is DEGREE for geographic spatial reference systems and NULL for non-geographic spatial reference systems.

The angular unit of measure must be non-NULL for geographic spatial reference systems and it must be NULL for non-geographic spatial reference systems.

To add predefined units of measure to the database, use the sa_install_feature system procedure.

To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

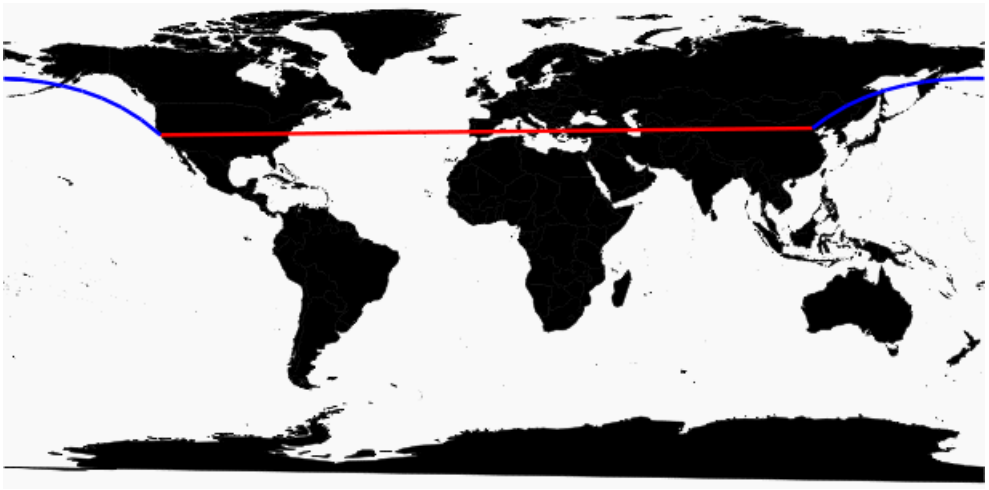
TYPE clause

Use the TYPE clause to control how the SRS interprets lines between points. For geographic spatial reference systems, the TYPE clause can specify either ROUND EARTH (the default) or PLANAR. The ROUND EARTH model interprets lines between points as great elliptic arcs. Given two points on the surface of the Earth, a plane is selected that intersects the two points and the center of the Earth. This plane intersects the Earth, and the line between the two points is the shortest distance along this intersection.

For two points that lie directly opposite each other, there is not a single unique plane that intersects the two points and the center of the Earth. Line segments connecting these antipodal points are not valid and give an error in the ROUND EARTH model.

The ROUND EARTH model treats the Earth as a spheroid and selects lines that follow the curvature of the Earth. In some cases, it may be necessary to use a planar model where a line between two points is interpreted as a straight line in the equirectangular projection where x=long, y=lat.

In the following example, the blue line shows the line interpretation used in the ROUND EARTH model and the red line shows the corresponding PLANAR model.



The PLANAR model may be used to match spatial interpretation used in other software products. The PLANAR model may also be useful because there are some limitations for methods that are not supported in the ROUND EARTH model (such as ST_Area, ST_ConvexHull) and some are partially supported (ST_Distance only supported between point geometries). Geometries based on circularstrings are not supported in ROUND EARTH spatial reference systems.

For non-geographic SRSs, the type must be PLANAR (and that is the default if the TYPE clause is not specified and either the DEFINITION clause is not specified or it uses a non-geographic definition).

COORDINATE clause

Use this clause to specify the bounds on the spatial reference system's dimensions. `coordinate-name` is the name of the coordinate system used by the spatial reference system. For non-geographic coordinate systems, `coordinate-name` can be x, y, or m. For geographic coordinate systems, `coordinate-name` can be LATITUDE, LONGITUDE, z, or m.

Specify UNBOUNDED to place no bounds on the dimensions. Use the BETWEEN clause to set low and high bounds.

The X and Y coordinates must have associated bounds. For geographic spatial reference systems, the longitude coordinate is bounded between -180 and 180 degrees and the latitude coordinate is bounded between -90 and 90 degrees by default the unless COORDINATE clause overrides these settings. For non-geographic spatial reference systems, the CREATE statement must specify bounds for both X and Y coordinates.

LATITUDE and LONGITUDE are used for geographic coordinate systems. The bounds for LATITUDE and LONGITUDE default to the entire Earth, if not specified.

ELLIPSOID clause

Use the ellipsoid clause to specify the values to use for representing the Earth as an ellipsoid for spatial reference systems of type ROUND EARTH. If the DEFINITION clause is present, it can specify ellipsoid definition. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

The Earth is not a perfect sphere because the rotation of the Earth causes a flattening so that the distance from the center of the Earth to the North or South pole is less than the distance from the center to the equator. For this reason, the Earth is modeled as an ellipsoid with different values for the semi-major axis (distance from center to equator) and semi-minor axis (distance from center to the pole). It is most common to define an ellipsoid using the semi-major axis and the inverse flattening, but it can instead be

specified using the semi-minor axis (for example, this approach must be used when a perfect sphere is used to approximate the Earth). The semi-major and semi-minor axes are defined in the linear units of the spatial reference system, and the inverse flattening (1/f) is a ratio:

```
1/f = (semi-major-axis) / (semi-major-axis - semi-minor-axis)
```

SQL Anywhere uses the ellipsoid definition when computing distance in geographic spatial reference systems.

The ellipsoid must be defined for geographic spatial reference systems (either in the DEFINITION clause or the ELLIPSOID clause), and it must not be specified for non-geographic spatial reference systems.

SNAP TO GRID clause

For flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid SQL Anywhere uses when performing calculations. By default, SQL Anywhere selects a grid size so that 12 significant digits can be stored at all points in the space bounds for X and Y. For example, if a spatial reference system bounds X between -180 and 180 and Y between -90 and 90, then a grid size of 0.000000001 (1E-9) is selected.

`grid-size` must be large enough so that points snapped to the grid can be represented with equal precision at all points in the bounded space. If `grid-size` is too small, the server reports an error.

When set to 0, no snapping to grid is performed.

For round-Earth spatial reference systems, SNAP TO GRID must be set to 0.

Specify SNAP TO GRID DEFAULT to set the grid size to the default that the database server would use.

TOLERANCE clause

For flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points. If the distance between two points is less than `tolerance-distance`, the two points are considered equal. Setting `tolerance-distance` allows you to control the tolerance for imprecision in the input data or limited internal precision. By default, `tolerance-distance` is set to be equal to `grid-size`.

When set to 0, two points must be exactly equal to be considered equal.

For round-Earth spatial reference systems, TOLERANCE must be set to 0.

POLYGON FORMAT clause

Internally, SQL Anywhere interprets polygons by looking at the orientation of the constituent rings. As one travels a ring in the order of the defined points, the inside of the polygon is on the left side of the ring. The same rules are applied in PLANAR and ROUND EARTH spatial reference systems.

The interpretation used by SQL Anywhere is a common but not universal interpretation. Some products use the exact opposite orientation, and some products do not rely on ring orientation to interpret polygons. The POLYGON FORMAT clause can be used to select a polygon interpretation that matches the input data, as needed. The following values are supported:

'CounterClockwise'

The input follows SQL Anywhere's internal interpretation: the inside of the polygon is on the left side while following ring orientation.

'Clockwise'

The input follows the opposite of SQL Anywhere's approach: the inside of the polygon is on the right side while following ring orientation.

'EvenOdd'

EvenOdd is the default format. With EvenOdd, the orientation of rings is ignored and the inside of the polygon is instead determined by looking at the nesting of the rings, with the exterior ring being the largest ring and interior rings being smaller rings inside this ring. A ray is traced from a point within the rings and radiating outward crossing all rings. If the number the ring being crossed is an even number, it is an outer ring. If it is odd, it is an inner ring.

STORAGE FORMAT clause

When you insert spatial data into the database from an external format (such as WKT or WKB), the database server normalizes the data to improve the performance and semantics of spatial operations. The normalized representation may differ from the original representation (for example, in the orientation of polygon rings or the precision stored in individual coordinates). While spatial equality is maintained after the normalization, some original input characteristics may not be reproducible, such as precision and ring orientation. In some cases you may want to store the original representation, either exclusively, or in addition to the normalized representation.

To control what is stored, specify the STORAGE FORMAT clause followed by one of the following values:

'Internal'

SQL Anywhere stores only the normalized representation. Specify this value when the original input characteristics do not need to be reproduced. This is the default for planar spatial reference systems (TYPE PLANAR).

> **i** Note
>
> If you are using MobiLink to synchronize your spatial data, you should specify **Mixed**. MobiLink tests for equality during synchronization, which requires the data in its original format.

'Original'

SQL Anywhere stores only the original representation. The original input characteristics can be reproduced, but all operations on the stored values must repeat normalization steps, possibly slowing down operations on the data.

'Mixed'

SQL Anywhere stores the internal version and, if it is different from the original version, it stores the original version as well. By storing both versions, the original representation characteristics can be reproduced and operations on stored values do not need to repeat normalization steps. However, storage requirements may increase significantly because potentially two representations are being stored for each geometry.

Mixed is the default format for round-Earth spatial reference systems (TYPE ROUND EARTH).

Remarks

For a geographic spatial reference system, you can specify both a LINEAR *and* an ANGULAR unit of measure; otherwise for non-geographic, you specify only a LINEAR unit of measure. The LINEAR unit of measure is used for computing distance between points and areas. The ANGULAR unit of measure tells how the angular

latitude/longitude are interpreted and is NULL for projected coordinate systems, non-NULL for geographic coordinate systems.

All derived geometries returned by operations are normalized.

When working with data that is being synchronized with a non-SQL Anywhere database, STORAGE FORMAT should be set to either 'Original' or 'Mixed' so that the original characteristics of the data can be preserved.

You cannot alter a spatial reference system (OR REPLACE) if there is existing data that references it. For example, if you have a column declared as ST_Point(SRID=8743), you cannot alter the spatial reference system with SRID 8743. This is because many spatial reference system attributes, such as storage format, impact the storage format of the data. If you have data that references the SRID, create a new spatial reference system and transform the data to the new SRID.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY SPATIAL OBJECT or CREATE ANY OBJECT system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a spatial reference system named mySpatialRS:

```
CREATE SPATIAL REFERENCE SYSTEM  "mySpatialRS"
IDENTIFIED BY 1000026980
LINEAR UNIT OF MEASURE "metre"
TYPE PLANAR
COORDINATE X BETWEEN 171266.736269555 AND 831044.757769222
COORDINATE Y BETWEEN 524881.608973277 AND 691571.125115319
DEFINITION 'PROJCS["NAD83 / Kentucky South",
GEOGCS["NAD83",
DATUM["North_American_Datum_1983",
```

```
SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],
AUTHORITY["EPSG","6269"]],
PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
UNIT["degree",0.01745329251994328,AUTHORITY["EPSG","9122"]],
AUTHORITY["EPSG","4269"]],
UNIT["metre",1,AUTHORITY["EPSG","9001"]],
PROJECTION["Lambert_Conformal_Conic_2SP"],
PARAMETER["standard_parallel_1",37.93333333333333],
PARAMETER["standard_parallel_2",36.73333333333333],
PARAMETER["latitude_of_origin",36.33333333333334],
PARAMETER["central_meridian",-85.75],
PARAMETER["false_easting",500000],
PARAMETER["false_northing",500000],
AUTHORITY["EPSG","26980"],
AXIS["X",EAST],
AXIS["Y",NORTH]]'
TRANSFORM DEFINITION '+proj=lcc
+lat_1=37.93333333333333+lat_2=36.73333333333333+lat_0=36.33333333333334+lon_0=-8
5.75+x_0=500000+y_0=500000+ellps=GRS80+datum=NAD83+units=m+no_defs';
```

Related Information

Spatial Reference Systems (SRS) and Spatial Reference Identifiers (SRID)
Spatial Data
sa_install_feature System Procedure [page 1625]
CREATE SPATIAL UNIT OF MEASURE Statement [page 989]
ST_UNITS_OF_MEASURE Consolidated View [page 1979]
ST_SPATIAL_REFERENCE_SYSTEMS Consolidated View [page 1976]
ALTER SPATIAL REFERENCE SYSTEM Statement [page 730]
sa_install_feature System Procedure [page 1625]

1.4.4.90 CREATE SPATIAL UNIT OF MEASURE Statement

Creates or replaces a spatial unit of measurement.

Syntax

```
CREATE [ OR REPLACE ] SPATIAL UNIT OF MEASURE identifier
TYPE {LINEAR | ANGULAR }
[ CONVERT USING number ][ IF NOT EXISTS ]
```

Parameters

OR REPLACE clause

Including the OR REPLACE creates a new spatial unit of measure, or replaces an existing spatial unit of measure with the same name. This clause preserves existing privileges. An error is returned if you attempt to replace a spatial unit that is already in use.

You cannot specify the IF NOT EXISTS clause if you specify this clause.

TYPE clause

Defines whether the unit of measure is used for angles (ANGULAR) or distances (LINEAR).

CONVERT USING

The conversion factor for the spatial unit relative to the base unit. For linear units, the base unit is METRE. For angular units, the base unit is RADIAN.

IF NOT EXISTS

Creates the spatial unit of measure if it does not exist and does not return any message. You cannot specify this clause if you specify the OR REPLACE clause.

Remarks

The CONVERT USING clause is used to define how to convert a measurement in the defined unit of measure to the base unit of measure (radians or meters). The measurement is multiplied by the supplied conversion factor to get a value in the base unit of measure. For example, a measurement of 512 millimeters would be multiplied by a conversion factor of 0.001 to get a measurement of 0.512 metres.

Spatial reference systems always include a linear unit of measure to be used when calculating distances (ST_Distance or ST_Length), or area. For example, if the linear unit of measure for a spatial reference system is miles, then the area unit used is square miles. In some cases, spatial methods accept an optional parameter that specifies the linear unit of measure to use. For example, if the linear unit of measure for a spatial reference system is in miles, you could retrieve the distance between two geometries in meters by using the optional parameter 'metre'.

For projected coordinate systems, the X and Y coordinates are specified in the linear unit of the spatial reference system. For geographic coordinate systems, the latitude and longitude are specified in the angular units of measure associated with the spatial reference system. In many cases, this angular unit of measure is degrees but any valid angular unit of measure can be used.

You can use the sa_install_feature system procedure to add predefined units of measure to your database.

Privileges

You must have the MANAGE ANY SPATIAL OBJECT or CREATE ANY OBJECT system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a spatial unit of measure named Test.

```
CREATE SPATIAL UNIT OF MEASURE Test
TYPE LINEAR
CONVERT USING 15;
```

Related Information

Spatial Data
sa_install_feature System Procedure [page 1625]
DROP SPATIAL UNIT OF MEASURE Statement [page 1124]

1.4.4.91 CREATE STATISTICS Statement

Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

Syntax

```
CREATE STATISTICS [ ON ] object-name [ ( column-list ) ]
```

```
object-name :
table-name | materialized-view-name | temp-table-name
```

Remarks

The CREATE STATISTICS statement recreates the column statistics that the database server uses to optimize database queries, and can be performed on base tables, materialized views, local temporary tables, and global temporary tables. You cannot create statistics on proxy tables. Column statistics include histograms, which reflect the distribution of data in the database for the specified columns. By default, column statistics are automatically created for tables with five or more rows.

In rare circumstances, when your database queries are very variable, and when data distribution is not uniform or the data is changing frequently, you can improve performance by executing the CREATE STATISTICS statement on a table or column.

When executing, the CREATE STATISTICS statement updates existing column statistics regardless of the size of the table, unless the table is empty, in which case nothing is done. If column statistics exist for an empty table, they remain unchanged by the CREATE STATISTICS statement. To remove column statistics for an empty table, execute the DROP STATISTICS statement.

The process of executing CREATE STATISTICS performs a complete scan of the table. For this reason, careful consideration should be made before executing a CREATE STATISTICS statement.

If you drop statistics, it is recommended that you recreate them using the CREATE STATISTICS statement. Without statistics, the optimizer can generate inefficient data access plans, causing poor database performance.

The CREATE STATISTICS and DROP STATISTICS statements do not require or benefit from a commit with regards to the statistics, and only DROP STATISTICS has a commit as a side effect. However, the commit does not save changes to statistics (the statistics governor looks after that). Thus, while DROP STATISTICS does have a side effect of an automatic commit, it commits everything except for the dropping of the statistics.

Privileges

You must be the table owner, or have the MANAGE ANY STATISTICS or CREATE ANY OBJECT system privilege.

Side Effects

Execution plans may change.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement updates the column statistics for the ProductID column of the SalesOrderItems table:

```
CREATE STATISTICS ON GROUPO.SalesOrderItems ( ProductID );
```

Related Information

Optimizer Estimates and Statistics
DROP STATISTICS Statement [page 1126]
LOAD TABLE Statement [page 1247]
SYSCOLSTAT System View [page 1902]
Histogram Utility (dbhist)
sa_get_histogram System Procedure [page 1601]

1.4.4.92 CREATE SUBSCRIPTION Statement [SQL Remote]

Creates a subscription for a remote user to a publication.

Syntax

```
CREATE SUBSCRIPTION
TO publication-name [ ( subscription-value ) ]
FOR subscriber-id
```

```
publication-name : identifier
```

```
subscription-value : string
```

```
subscriber-id : string
```

Parameters

publication-name

The name of the publication to which the user is being subscribed. This can include the owner of the publication.

subscription-value

A string that is compared to the subscription expression of the publication. The subscriber receives all rows for which the subscription expression matches the subscription value.

You can specify a variable name for `subscription-value`. If the variable is NULL, the value behaves as if an empty string was specified.

subscriber-id

The user ID of the subscriber to the publication. At the consolidated database, when you create a subscription for a remote user, the remote user must have been granted REMOTE privilege. At the remote database, when you create a subscription for the consolidated user, that user must have been granted CONSOLIDATED privilege.

Remarks

In SQL Remote, publications and subscriptions are two-way relationships. If you create a subscription for a remote user to a publication on a consolidated database, you should also create a subscription for the consolidated user to a publication on the remote database. By default, the Extraction utility (dbxtract) and the *Extract Database Wizard* grant the appropriate PUBLISH and CONSOLIDATE privilege to users in the remote databases.

If *subscription-value* is supplied, it is matched against each SUBSCRIBE BY expression in the publication. The subscriber receives all rows for which the value of the expression is equal to the supplied string.

> **i** Note
>
> For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a subscription for the user p_chin to the publication pub_sales. The subscriber receives all rows for which the subscription expression has a value of Eastern.

```
CREATE SUBSCRIPTION
```

```
TO pub_sales ( 'Eastern' )
FOR p_chin;
```

The following statement creates a subscription for user name Sam_Singer to the CustomerPub publication, which was created using a WHERE clause:

```
CREATE SUBSCRIPTION
  TO CustomerPub
  FOR Sam_Singer;
```

The following statement creates a subscription for user name Sam_Singer to the PubOrders publication, defined with a subscription expression SalesRepresentative, requesting the rows for Sam Singer's own sales:

```
CREATE SUBSCRIPTION
  TO PubOrders ( '856' )
  FOR Sam_Singer;
```

Example

The following statement creates a variable for the value 'ny':

```
CREATE VARIABLE @state LONG VARCHAR ;
SET @state = 'ny' ;
```

The following statement creates a publication called pub_customer:

```
CREATE PUBLICATION pub_customer (
TABLE DBA>Customer ( ID, company_name, City, State )
SUBSCRIBE BY State ) ;
```

The following statement creates a subscription for user name Sam_Singer

```
CREATE SUBSCRIPTION
   TO pub_customer ( @dan )
   FOR Sam_Singer ;
```

The subscriber receives all rows for which the subscription expression has a value of the variable @dan, which is set to NY state.

Related Information

Subscriptions
Publishing Only Some Rows in a Table (SQL Central)
Replicating the Primary Key Pool (SQL)
Publishing Only Some Rows Using the SUBSCRIBE BY Clause (SQL Central)
Publishing Only Some Rows Using a WHERE Clause (SQL Central)
DROP SUBSCRIPTION Statement [SQL Remote] [page 1128]
GRANT REMOTE Statement [SQL Remote] [page 1208]
GRANT CONSOLIDATE Statement [SQL Remote] [page 1199]

SYNCHRONIZE SUBSCRIPTION Statement [SQL Remote] [page 1438]
START SUBSCRIPTION Statement [SQL Remote] [page 1416]
GRANT PUBLISH Statement [SQL Remote] [page 1206]
SYSSUBSCRIPTION System View [page 1947]

1.4.4.93 CREATE SYNCHRONIZATION PROFILE Statement [MobiLink]

Creates a SQL Anywhere synchronization profile.

Syntax

```
CREATE [ OR REPLACE ] SYNCHRONIZATION PROFILE name string
```

Parameters

OR REPLACE clause

Specifying CREATE OR REPLACE SYNCHRONIZATION PROFILE replaces the definition of the named synchronization profile if it already exists.

name

Specifies the name of the synchronization profile to create. Each profile must have a unique name.

string

Specify a valid option string as described below. Option strings are specified as semicolon-delimited lists of elements of the form `option-name=option-value`. For example `subscription=s1;verbosity=high`.

Remarks

Synchronization profiles are named collections of synchronization options that can be used to control synchronization.

For options that take a Boolean value, setting the value to TRUE is equivalent to specifying the corresponding option on the command line.

The following values can be used to specify TRUE: TRUE, ON, 1, YES.

The following values can be used to specify FALSE: FALSE, OFF, 0, NO.

When setting extended options, use the following syntax:

```
CREATE SYNCHRONIZATION PROFILE myprofile
's=mysub;e={ctp=tcpip;adr=''host=localhost;port=2439''}'
```

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

MobiLink Synchronization Profiles
ALTER SYNCHRONIZATION PROFILE Statement [MobiLink] [page 736]
DROP SYNCHRONIZATION PROFILE Statement [MobiLink] [page 1129]
SYNCHRONIZE Statement [MobiLink] [page 1433]

1.4.4.94 CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

Syntax

```
CREATE SYNCHRONIZATION SUBSCRIPTION [ subscription-name ]
TO publication-name
[ FOR ml-username, ... ]
[ TYPE network-protocol ]
[ ADDRESS protocol-options ]
[ OPTION option=value, ... ]
[ SCRIPT VERSION script-version ]
```

```
subscription-name : identifier
```

```
ml-username : identifier
```

```
network-protocol : http | https | tls | tcpip
```

```
protocol-options : string

value : string | integer

script-version : string
```

Parameters

subscription-name

A unique name that you can use to identify this subscription. It is strongly recommended that you name all your subscriptions.

TO clause

This clause specifies the name of a publication.

FOR clause

This clause specifies one or more MobiLink user names. If you specify more than one user name, a separate subscription is created for each user. If you specify a subscription name, only one MobiLink user name can be specified.

`ml-username` is a user who is authorized to synchronize with the MobiLink server.

Omit the FOR clause to set the protocol type, protocol options, and extended options for a publication. If the FOR clause is omitted, you cannot specify a subscription name or use the SCRIPT VERSION clause.

TYPE clause

This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause

This clause specifies network protocol options such as the location of the MobiLink server. Multiple options must be separated by semicolons.

OPTION clause

This clause allows you to set extended options for the subscription. If no FOR clause is provided, the extended options act as default settings for the publication.

SCRIPT VERSION clause

This clause specifies the script version to use during synchronization. Typically, you must specify a new script version for each schema change you implement.

You cannot use the SCRIPT VERSION clause if the FOR clause is omitted.

Remarks

If no `subscription-name` is specified, a unique name is generated. The generated subscription name is the same as the publication name, provided it is unique. Otherwise, a unique name is formed by adding a number to the end of the publication name, for example, pub001, pub002, and so on.

The `network-protocol`, `protocol-options`, and `option` values can be set in several places.

This statement causes options and other information to be stored in the ISYSSYNC system table. Anyone with privileges to view data in the SYSSYNC system view can view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

You must have exclusive access to all tables referenced in the publication to execute the statement.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a subscription named sales between the MobiLink user SSinger and the publication called sales_publication owned by `user`. When the subscription is synchronized, the script version sales_v1 is used and tables are locked in exclusive mode:

```
CREATE SYNCHRONIZATION SUBSCRIPTION sales
TO user.sales_publication
FOR SSinger
OPTION locktables='exclusive'
SCRIPT VERSION 'sales_v1'
```

The following example omits the FOR clause and stores settings for the publication called sales_publication:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO user.sales_publication
ADDRESS 'host=test.internal;port=2439'
OPTION locktables=exclusive';
```

Related Information

Synchronization Subscription Creation
MobiLink Users in a Synchronization System
MobiLink SQL Anywhere Client Extended Options
How dbmlsync Resolves Conflicting Options
Script Versions
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 737]
DROP SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 1131]
MobiLink Client Network Protocol Options
CommunicationType (ctp) Extended Option
SYSSYNC System View [page 1948]
MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax

1.4.4.95 CREATE SYNCHRONIZATION USER Statement [MobiLink]

Creates a MobiLink user in a SQL Anywhere remote database.

Syntax

```
CREATE SYNCHRONIZATION USER ml-username
[ TYPE network-protocol ]
[ ADDRESS protocol-options ]
[ OPTION option=value, ... ]
```

```
ml-username : identifier
```

```
network-protocol :
tcpip
| http
| https
| tls
```

```
protocol-options : string
```

```
value : string | integer
```

Parameters

ml_username

A name identifying a MobiLink user.

TYPE clause

This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause

This clause specifies `protocol-options` in the form `keyword=value`, separated by semicolons. Which settings you supply depends on the communication protocol you are using (TCP/IP, TLS, HTTP, or HTTPS).

OPTION clause

The OPTION clause allows you to set extended options using `option=value` in a comma-separated list.

The values for each option cannot contain equal signs or semicolons. The database server accepts any option that you enter without checking for its validity. Therefore, if you misspell an option or enter an invalid value, no error message appears until you run the dbmlsync command to perform synchronization.

Options set for a synchronization user can be overridden in individual subscriptions or on the dbmlsync command line.

The `network-protocol`, `protocol-options`, and `option` settings can be overridden in other statements and contexts.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Anyone with the correct privileges to view the SYSSYNC system view can view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a MobiLink user named SSinger, who synchronizes over TCP/IP with a server computer named mlserver.mycompany.com using the password Sam. The use of a password in the user definition is *not* secure.

```
CREATE SYNCHRONIZATION USER SSinger
TYPE http
ADDRESS 'host=mlserver.mycompany.com'
OPTION MobiLinkPwd='Sam';
```

Related Information

MobiLink Client/Server Communications Encryption
MobiLink Users in a Synchronization System
MobiLink SQL Anywhere Client Extended Options
How dbmlsync Resolves Conflicting Options
ALTER SYNCHRONIZATION USER Statement [MobiLink] [page 740]
DROP SYNCHRONIZATION USER Statement [MobiLink] [page 1132]
CommunicationType (ctp) Extended Option
SYSSYNC System View [page 1948]
MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax
MobiLink Client Network Protocol Options

1.4.4.96 CREATE TABLE Statement

Creates a new table in the database and, optionally, creates a table on a remote server.

Syntax

```
CREATE [ OR REPLACE ] TABLE [ IF NOT EXISTS ] [owner.]table-name
[ ( { column-definition | table-constraint | pct-free | like-clause }, ... ) |
as-clause | like-clause ]
[ { IN | ON } dbspace-name ]
[ ENCRYPTED ]
[ WITH [ NO ] DATA ]
[ AT location-string [ ESCAPE CHARACTER character ] ]
```

```
CREATE [ OR REPLACE ] GLOBAL TEMPORARY TABLE [ IF NOT EXISTS ] [owner.]table-
name
[ ( { column-definition | table-constraint | pct-free | like-clause }, ... ) |
as-clause | like-clause ]
[ ON COMMIT { DELETE | PRESERVE } ROWS
  | NOT TRANSACTIONAL ]
[ SHARE BY ALL ]
```

```
CREATE LOCAL TEMPORARY TABLE [ IF NOT EXISTS ] table-name
```

```
[ ( { column-definition | table-constraint | pct-free | like-clause }, ... ) |
as-clause | like-clause ]
[ ON COMMIT { DELETE | PRESERVE } ROWS
  | NOT TRANSACTIONAL ]
```

```
column-definition :
column-name data-type
[ COMPRESSED ]
[ INLINE { inline-length | USE DEFAULT } ]
[ PREFIX { prefix-length | USE DEFAULT } ]
[ [ NO ] INDEX ]
[ [ NOT ] NULL ]
[ DEFAULT default-value | IDENTITY | COMPUTE ( expression ) ]
[ column-constraint ... ]
[ table-element ... ]
```

```
default-value :
  special-value
| string
| global-variable
| [ - ] number
| ( constant-expression )
| ( sequence-expression )
| built-in-function( constant-expression )
| AUTOINCREMENT
| GLOBAL AUTOINCREMENT [ ( partition-size ) ]
```

```
special-value :
  CURRENT DATABASE
| CURRENT DATE
| CURRENT TIME
| [ CURRENT ] TIMESTAMP
| CURRENT PUBLISHER
| CURRENT REMOTE USER
| [ CURRENT ] USER
| [ CURRENT ] UTC TIMESTAMP
| EXECUTING USER
| INVOKING USER
| LAST USER
| NULL
| PROCEDURE OWNER
| SESSION USER
```

```
column-constraint :
  [ CONSTRAINT constraint-name ]
  { UNIQUE [ CLUSTERED | NONCLUSTERED ]
  | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ]
  | REFERENCES table-name [ ( column-name ) ]
     [ MATCH [ UNIQUE ] { SIMPLE | FULL } ]
     [ action-list ] [ CLUSTERED | NONCLUSTERED ]
  | CHECK ( condition )
  }
```

```
table-element :
 { column-definition
 | table-constraint-definition
 | like-clause }
```

```
table-constraint :
  [ CONSTRAINT constraint-name ]
  {   UNIQUE [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC | DESC ], ... )
```

```
    | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC |
DESC ], ... )
    | CHECK ( condition )
    | foreign-key-constraint
  }

foreign-key-constraint :
[ NOT NULL ] FOREIGN KEY [ role-name ]
   [ ( column-name [ ASC | DESC ], ... ) ]
   REFERENCES table-name
   [ ( column-name, ... ) ]
   [ MATCH [ UNIQUE] { SIMPLE | FULL } ]
   [ action-list ] [ CHECK ON COMMIT ] [ CLUSTERED | NONCLUSTERED ] [ FOR
OLAP WORKLOAD ]

action-list :
[ ON UPDATE action ]
[ ON DELETE action ]

action :
CASCADE
| SET NULL
| SET DEFAULT
| RESTRICT

pct-free : PCTFREE percent-free-space

percent-free-space : integer

as-clause :
[ (column-name, ... ) ] AS ( select-statement )

like-clause :
LIKE source-table [ like-option [ ... ] ]

like-option :
{ INCLUDING | EXCLUDING } option

option :
  ALL
| IDENTITY
| DEFAULTS
| CONSTRAINTS
| INDEXES
| PRIMARY KEY
| FOREIGN KEYS
| COMMENTS
| STORAGE

location-string :
 remote-server-name.[db-name].[owner].object-name
| remote-server-name;[db-name];[owner];object-name
```

Parameters

OR REPLACE clause

The table definition is replaced if it already exists, and an error is not returned. This clause cannot be specified if IF NOT EXISTS is specified.

IF NOT EXISTS clause

No changes are made if the named table already exists, and an error is not returned. This clause cannot be specified if OR REPLACE is specified.

AS clause

Use this clause to clone a table based on a SELECT statement.

LIKE clause

Use this clause to clone a table from another table or view. The default for `like-option` is to exclude (EXCLUDING) all options. If `source-table` is a view, then `like-option` is ignored if it is not applicable.

Specify INCLUDE to include any of the following column attributes from the original table in the new table:

ALL Include all of the column attributes.
IDENTITY Include default autoincrement information.
DEFAULTS Include default value expressions, excluding autoincrement.
CONSTRAINTS Include column and table check and unique constraints.
INDEXES Include indexes, excluding primary and foreign keys.
PRIMARY KEY Include the primary key.
FOREIGN KEY Include foreign keys.
COMMENTS Include comments on columns.
STORAGE Include the compression setting.

IN clause

Use this clause to specify the dbspace in which the base table is located. If this clause is not specified, then the base table is created in the dbspace specified by the default_dbspace option.

Temporary tables can only be created in the temporary dbspace. If you are creating a GLOBAL TEMPORARY table, and specify IN, the table is created in the temporary dbspace. If you specify a user-defined dbspace, an error is returned.

ENCRYPTED clause

The encrypted clause specifies that the table should be encrypted. You must enable table encryption when you create a database if you plan to encrypt tables. The table is encrypted using the encryption key and algorithm specified at database creation time.

WITH [NO] DATA clause Use this clause only when you have specified the AS clause to specify whether or not to copy data from the original table into the new table. By default, data is copied.

AT clause

Create a remote table on a different server specified by `location-string`, and a proxy table on the current database that maps to the remote table. The AT clause supports the semicolon (;) as a field delimiter in `location-string`. If no semicolon is present, a period is the field delimiter. This syntax allows file names and extensions to be used in the database and owner fields.

For example, the following statement maps the table proxy_Customers to a new table Customers in a Microsoft Access database called MyAccessDB:

```
CREATE TABLE proxy_Customers
(
    ID            INTEGER CONSTRAINT PRIMARY KEY,
    ClientName    TEXT,
    ClientAddress TEXT,
    Telephone     TEXT
)
AT 'MyAccessDB;;;Customers';
```

The string in the AT clause can also contain local or global variable names enclosed in braces ({variable-name}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains `'access;{@myfile};;a1'` indicates that `@myfile` is a SQL variable and that the current contents of the `@myfile` variable should be substituted when the proxy table is created.

Foreign key definitions are ignored on remote tables. Foreign key definitions on local tables that refer to remote tables are also ignored. Primary key definitions are sent to the remote server if the database server supports primary keys.

When you create a proxy table by using either the CREATE TABLE or the CREATE EXISTING statement, the AT clause includes a location string that is comprised of the following parts:

- The name of the remote server
- The remote catalog
- The remote owner or schema
- The remote table name

`location-string` can include variable names that are expanded when the database server evaluates the location string. Variable names within `location-string` must be encapsulated within braces.

ESCAPE CHARACTER clause

The ESCAPE CHARACTER clause allows you to escape a character in a remote server name, catalog name, owner name, schema name, or table name. For example, if the object name contains a character such as period, semicolon, and a brace, it must be escaped by specifying the ESCAPE CHARACTER clause.

`character` can be any single byte character.

GLOBAL TEMPORARY clause

The GLOBAL TEMPORARY clause identifies the table as global and temporary. The table is global since the schema persists and is available to all connections. The table is temporary since the rows in the table are not persisted. The rows in the table may or may not be shared depending on the SHARE BY ALL clause.

LOCAL TEMPORARY clause

The LOCAL TEMPORARY clause identifies the table as local and temporary. The table is local since the schema is not shared with other connections. The table is temporary since the rows in the table are not persisted. The rows in the table are not shared with other connections.

ON COMMIT clause

The ON COMMIT clause is allowed when creating a global or local temporary table. By default, the rows of a temporary table are deleted on COMMIT. For a global temporary table, if the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

NOT TRANSACTIONAL clause

The NOT TRANSACTIONAL clause is allowed when creating a global or local temporary table. A temporary table created using NOT TRANSACTIONAL is not affected by either COMMIT or ROLLBACK. For a global temporary table, ilf the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

SHARE BY ALL clause

The SHARE BY ALL clause enables the sharing of the rows of a global temporary table by all connections to the database. If the SHARE BY ALL clause is not specified, then only the table schema is shared and each connection has its own private set of rows which can persist for the duration of the connection. If the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified and the table rows can persist until the database is shut down. The SHARE BY ALL clause is not permitted for local temporary tables.

column-definition

Define a column in the table. The following are part of column definitions.

column-name

The column name is an identifier. Two columns in the same table cannot have the same name.

data-type

The type of data stored in the column. Define the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in another table or view. When you specify the %TYPE attribute, other attributes on the object referenced in the %TYPE specification, such as default values, constraints, and whether NULLs are allowed, are not part of the definition that is inherited and must be specified separately.

The use of %TYPE or %ROWTYPE is not allowed for variable or temporary objects.

COMPRESSED clause

Compress the column.

INLINE and PREFIX clauses

The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. In this context, the term BLOB refers to both character and binary column values of any length.

BLOBs that are shorter than or equal to the INLINE value are stored inline, that is, within the row.

BLOBs that are longer than the value specified by the INLINE clause are stored outside the row in table extension pages. In this case, a prefix of the BLOB may be stored inline. The prefix is composed from the leading bytes of the BLOB. Use the PREFIX clause to specify the length of the prefix. A PREFIX value of 0 means that no prefix will be stored. The PREFIX value cannot exceed the INLINE value. The PREFIX clause can affect the performance of requests that use the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The data that is stored inline is not compressed. If a prefix exists and it indicates that the BLOB value will not satisfy the request, then no decompression or the data stored outside the row in table extension pages is necessary.

If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.

- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

  > **i Note**
  >
  > It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set INLINE to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

  If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the INLINE nor PREFIX value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an INLINE value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

INDEX and NO INDEX clauses

When storing BLOBs (character or binary types only), specify INDEX to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify NO INDEX.

> **i Note**
>
> A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

NULL and NOT NULL clauses

If NULL is specified, NULL values are allowed in the column. This is the default behavior. This setting is controlled by the allow_nulls_by_default database option.

By default, columns declared as BIT are not nullable, but they can be explicitly made nullable.

If NOT NULL is specified, NULL values are not allowed.

If the column is part of a UNIQUE or PRIMARY KEY constraint, the column cannot contain NULL, even if NULL is specified.

DEFAULT clause

If a DEFAULT value is specified, it is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT value is specified, it is equivalent to DEFAULT NULL.

Following is a list of possible values for DEFAULT:

constant-expression

Constant expressions that do not reference database objects are allowed in a DEFAULT clause, so functions such as GETDATE or DATEADD can be used. If the expression is not a function or simple value, it must be enclosed in parentheses.

global-variable

A global variable.

sequence-expression

You can set DEFAULT to the current value or next value from a sequence in the database.

string

A string value.

AUTOINCREMENT

When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type.

On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, that value is inserted and then used as a starting point for subsequent inserts.

Deleting rows does not decrement the AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. After an explicit insert of a column value less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the next value to be generated exceeds the maximum value that can be stored in the column to which the AUTOINCREMENT is assigned, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated.

The next value to use for a column can be reset using the sa_reset_identity procedure.

GLOBAL AUTOINCREMENT

This default is intended for use when multiple databases are used in a MobiLink synchronization environment or SQL Remote replication.

This clause is similar to AUTOINCREMENT, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. The database server supplies default values in a database only from the partition uniquely identified by that database's number.

The partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size can be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

If the column is of type BIGINT or UNSIGNED BIGINT, the default partition size is 2^{32} = 4294967296; for columns of all other types, the default partition size is 2^{16} = 65536. Since these defaults may be inappropriate, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

When using this default, the value of the public option global_database_id in each database must be set to a unique, non-negative integer. This value uniquely identifies the database and indicates from which partition default values are to be assigned. The range of allowed values is $n\,p + 1$ to $p(n + 1)$,

where n is the value of the public option global_database_id and p is the partition size. For example, if you define the partition size to be 1000 and set global_database_id to 3, then the range is from 3001 to 4000.

If the previous value is less than p(n + 1), the next default value is one greater than the previous largest value in the column. If the column contains no values, the first default value is n p + 1. Default column values are not affected by values in the column outside the current partition; that is, by numbers less than np + 1 or greater than p(n + 1). Such values may be present if they have been replicated from another database via MobiLink or SQL Remote.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

GLOBAL AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the supply of values within the partition has been exhausted, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated. In this case, a new value of global_database_id should be assigned to the database to allow default values to be chosen from another partition. To detect that the supply of unused values is low and handle this condition, create an event of type GlobalAutoincrement.

Because the public option global_database_id cannot be set to a negative value, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

If the public option global_database_id is set to the default value of 2147483647, a NULL value is inserted into the column. If NULL values are not permitted, attempting to insert the row causes an error.

The next value to use for a column can be reset using the sa_reset_identity procedure.

special-value

You use one of several special values in the DEFAULT clause (for example, CURRENT DATE) including, but not limited to, the following special values.

[CURRENT] TIMESTAMP

Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the current date and time of day whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT TIMESTAMP instead of DEFAULT TIMESTAMP.

Columns declared with DEFAULT TIMESTAMP contain unique values, so that applications can detect near-simultaneous updates to the same row. If the current TIMESTAMP value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate TIMESTAMP values based on the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable @@dbts returns a TIMESTAMP value representing the last value generated for a column using DEFAULT TIMESTAMP.

When the database is using a simulated time zone, the simulated time zone is used to calculate these values.

[CURRENT] UTC TIMESTAMP

Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT UTC TIMESTAMP, a default value is provided for inserts, and the value is updated with the current Coordinated Universal Time (UTC) whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT UTC TIMESTAMP instead of DEFAULT UTC TIMESTAMP.

The behavior of this default is the same as TIMESTAMP and CURRENT TIMESTAMP except that the date and time of day is in Coordinated Universal Time (UTC).

The global variable @@dbts does not get updated when using CURRENT UTC TIMESTAMP.

LAST USER

LAST USER is the user ID of the user who last modified the row.

LAST USER can be used as a default value in columns with character data types.

On INSERT, this default has the same effect as CURRENT USER.

On UPDATE, if a column with a default of LAST USER is not explicitly modified, it is changed to the name of the current user.

When used with DEFAULT TIMESTAMP or DEFAULT UTC TIMESTAMP, a default of LAST USER can be used to record (in separate columns) both the user and the date and time a row was last changed.

IDENTITY clause

IDENTITY is a Transact-SQL-compatible alternative to using DEFAULT AUTOINCREMENT. A column defined with IDENTITY is implemented as DEFAULT AUTOINCREMENT.

column-constraint and table-constraint clauses

Column and table constraints help ensure the integrity of data in the database. If a statement would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are undone, and an error is reported. There are two classes of constraints that can be created: **check constraint**, and **referential integrity (RI) constraints**. Check constraints are used to specify conditions that must be satisfied by values of columns being put into the database. RI constraints establish a relationship between data in different tables that must be maintained in addition to specifying uniqueness requirements for data.

There are three types of RI constraints: primary key, foreign key, and unique constraint. When you create an RI constraint (primary key, foreign key or unique constraint), the database server enforces the constraint by implicitly creating an index on the columns that make up the key of the constraint. The index is created on the key for the constraint as specified. A key consists of an ordered list of columns and a sequencing of values (ASC/DESC) for each column.

Constraints can be specified on columns or tables. A column constraint refers to one column in a table, while a table constraint can refer to one or more columns in a table.

PRIMARY KEY clause

A primary key uniquely defines each row in the table. Primary keys comprise one or more columns. A table cannot have more than one primary key. In a `column-constraint` clause, specifying PRIMARY KEY indicates that the column is the primary key for the table. In a `table-constraint`, you use the

PRIMARY KEY clause to specify one or more columns that, when combined in the specified order, make up the primary key for the table.

The ordering of columns in a primary key need not match the respective ordinal numbers of the columns. That is, the columns in a primary key need not have the same physical order in the row. Additionally, you cannot specify duplicate column names.

When you create a primary key, an index for the key is automatically created. For a table constraint, you can specify the sequencing of values in the index by specifying ASC (ascending) or DESC (descending) for each column. You can also specify whether to cluster the index, using the CLUSTERED keyword.

Columns included in primary keys cannot allow NULL. Each row in the table has a unique primary key value.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for primary keys. Approximate numeric data types are subject to rounding errors after arithmetic operations.

Spatial columns cannot be included in a primary key.

FOREIGN KEY clause

A foreign key restricts the values for a set of columns to match the values in a primary key or a unique constraint of another table (the primary table). For example, a foreign key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the Customers table.

The foreign key column order does not need to reflect the order of columns in the table.

Duplicate column names are not allowed in the foreign key specification.

When you create a foreign key, an index for the key is automatically created. You can specify the sequencing of values in the index by specifying ASC (ascending) or DESC (descending) for each column. You can also specify whether to cluster the index, using the CLUSTERED keyword.

A global temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a global temporary table.

If you attempt to add a foreign key to a column that does not exist, that column is automatically created.

NOT NULL clause

Disallow NULLs in the foreign key columns. A NULL in a foreign key means that no row in the primary table corresponds to this row in the foreign table.

role-name clause

The role name is the name of the foreign key. The main function of the role name is to distinguish between two foreign keys to the same table. If no role name is specified, the role name is assigned as follows:

1. If there is no foreign key with a role name the same as the table name, the table name is assigned as the role name.
2. If the table name is already taken, the role name is the table name concatenated with a zero-padded three-digit number unique to the table.

REFERENCES clause

A foreign key constraint can be implemented using a REFERENCES column constraint (single column only) or a FOREIGN KEY table constraint, in which case the constraint can specify one or more columns. If you specify `column-name` in a REFERENCES column constraint, it must be a column in the primary table, must be subject to a unique constraint or primary key constraint, and that constraint must consist of only that one column. If you do not specify `column-name`, the foreign key column references the single primary key column of the primary table.

MATCH clause

The MATCH clause determines what is considered a match when using a multi-column foreign key by allowing you to regulate what constitutes an orphaned row versus what constitutes a violation of referential integrity. The MATCH clause also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

The following is a list of MATCH types you can specify.

MATCH [UNIQUE] SIMPLE

A match occurs for a row in the foreign key table if all the column values match the corresponding column values present in a row of the primary key table. A row is orphaned in the foreign key table if at least one column value in the foreign key is NULL.

MATCH SIMPLE is the default behavior.

If the UNIQUE keyword is specified, the referencing table can have only one match for non-NULL key values.

MATCH [UNIQUE] FULL

A match occurs for a row in the foreign key table if none of the values are NULL and the values match the corresponding column values in a row of the primary key table. A row is orphaned if all column values in the foreign key are NULL.

If the UNIQUE keyword is specified, the referencing table can have only one match for non-NULL key values.

UNIQUE clause

In a `column-constraint` clause, a UNIQUE constraint specifies that the values in the column must be unique. In a `table-constraint` clause, the UNIQUE constraint identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named column(s). A table can have more than one UNIQUE constraint.

A UNIQUE constraint is not the same as a unique index. Columns of a unique index are allowed to be NULL, while columns in a UNIQUE constraint are not. Also, a foreign key can reference either a primary key or a UNIQUE constraint, but cannot reference a unique index since a unique index can include multiple instances of NULL.

Columns in a UNIQUE constraint can be specified in any order. Additionally, you can specify the sequencing of values in the corresponding index that is automatically created, by specifying ASC (ascending) or DESC (descending) for each column. You cannot specify duplicate column names, however.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for columns with unique constraints. Approximate numeric data types are subject to rounding errors after arithmetic operations.

You can also specify whether to cluster the constraint, using the CLUSTERED keyword.

action clause (CASCADE, SET NULL, SET DEFAULT, RESTRICT)

The referential integrity action to take when updates and deletes would affect foreign keys.

CASCADE

For ON UPDATE, this action updates all foreign keys that reference the updated primary key to the new value. For ON DELETE, this action deletes all foreign key rows that reference the deleted primary key.

SET NULL

Sets all foreign keys that reference the modified primary key to NULL.

SET DEFAULT

Sets all foreign keys that reference the modified primary key to the default value for that column (as specified in the table definition).

RESTRICT

Generates an error and prevents the modification if an attempt to alter a referenced primary key value occurs. This is the default referential integrity action.

CHECK clause

This constraint allows arbitrary conditions to be verified. For example, a CHECK constraint could be used to ensure that a column called Sex only contains the values M or F.

If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

No row in a table is allowed to violate a CHECK constraint. If an INSERT or UPDATE statement would cause a row to violate the constraint, the operation is not permitted and the effects of the statement are undone. The change is rejected only if a CHECK constraint condition evaluates to FALSE, and the change is allowed if a CHECK constraint condition evaluates to TRUE or UNKNOWN.

COMPUTE clause

The COMPUTE clause is only for use in a `column-constraint` clause.

When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The COMPUTE clause is ignored for remote tables.

Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.

CHECK ON COMMIT clause

The CHECK ON COMMIT clause overrides the wait_for_commit database option, and causes the database server to wait for a COMMIT before checking RESTRICT actions on a foreign key. The CHECK ON COMMIT clause delays foreign key checking, but does not delay other actions such as CASCADE, SET NULL, SET DEFAULT, or check constraints.

FOR OLAP WORKLOAD clause

When you specify FOR OLAP WORKLOAD in the REFERENCES clause of a foreign key definition, the database server performs certain optimizations and gathers statistics on the key to help improve performance for OLAP workloads, particularly when the optimization_workload option is set to OLAP.

PCTFREE clause

Specifies the percentage of free space you want to reserve for each table page. The free space is used if rows increase in size when the data is updated. If there is no free space in a table page, every increase in the size of a row on that page requires the row to be split across multiple table pages, causing row fragmentation and possible performance degradation.

The value `percent-free-space` is an integer between 0 and 100. The former value specifies that no free space is to be left on each page, each page is to be fully packed. A high value causes each row to be inserted into a page by itself. If PCTFREE is not set, or is later dropped, the default PCTFREE value is applied according to the database page size (200 bytes for a 4 KB (and up) page size). The value for PCTFREE is stored in the ISYSTAB system table.

Remarks

The CREATE TABLE statement creates a new table. A table can be created for another user by specifying an owner.

If GLOBAL TEMPORARY is specified, the table is a temporary table. Otherwise, the table is a base table. Like a base table, the schema of the global temporary table is available to all connections. If ON COMMIT PRESERVE ROWS is specified, then the rows in the global temporary table persist while the connection persists, unless SHARE BY ALL is also specified, in which case the rows in the global temporary table persist until the database is shut down.

In a procedure, use the CREATE LOCAL TEMPORARY TABLE statement, instead of the DECLARE LOCAL TEMPORARY TABLE statement, when you want to create a table that persists after the procedure completes. Local temporary tables created using the CREATE LOCAL TEMPORARY TABLE statement remain until they are either explicitly dropped, or until the connection closes.

Tables created using CREATE LOCAL TEMPORARY TABLE do not appear in the SYSTABLE view of the system catalog.

Local temporary tables created in IF statements using CREATE LOCAL TEMPORARY TABLE also persist after the IF statement completes.

Two local temporary tables within the same scope cannot have the same owner and name. If you create a local temporary table with the same owner and name as a base table, the base table only becomes visible within the connection once the scope of the local temporary table ends. A connection cannot create a base table with the same owner and name as an existing temporary table.

The CREATE TABLE...LIKE syntax creates a new table based directly on the definitions of another table. You can also clone a table with additional columns, constraints, and LIKE clauses, or create a table using a SELECT statement.

Tables created by preceding the table name in a CREATE TABLE statement with a number sign (#) are declared temporary tables, which are available only in the current connection. Temporary tables created with the number sign (#) are identical to those created with the ON COMMIT PRESERVE ROWS clause.

If you execute the following set of SQL statements without semicolons to break up the statements, then this is treated as a T-SQL (Transact SQL) batch and the temporary table has scope limited to the batch.

```
CREATE TABLE #TEMP( C1 INT, C2 INT )
INSERT INTO #TEMP (SELECT 1,2)
```

```
SELECT * FROM #TEMP
```

Execution of a subsequent SELECT statement will return a table not found error.

If semicolons are used as follows then a Watcom SQL batch is assumed and the CREATE TABLE statement is interpreted as a CREATE LOCAL TEMPORARY TABLE statement which has connection scope.

```
CREATE TABLE #TEMP( C1 INT, C2 INT );
INSERT INTO #TEMP (SELECT 1,2);
SELECT * FROM #TEMP;
```

Execution of a subsequent SELECT statement will succeed.

Two local temporary tables within the same scope cannot have the same owner and name. If you create a local temporary table with the same owner and name as a base table, the base table only becomes visible within the connection once the scope of the local temporary table ends. A connection cannot create a base table with the same owner and name as an existing temporary table.

Columns allow NULLs by default. This setting can be controlled using the allow_nulls_by_default database option.

Privileges

Creation of LOCAL TEMPORARY tables does not require any privileges.

For base tables and GLOBAL TEMPORARY tables, you must have the CREATE TABLE system privilege to create tables owned by you. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create tables owned by others.

To create proxy tables owned by you, you must have the CREATE PROXY TABLE system privilege. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create proxy tables owned by others.

To replace an existing table, you must be the owner of the table, or have one of the following:

- CREATE ANY TABLE and DROP ANY TABLE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY TABLE system privileges.

Side Effects

Automatic commit (even when creating global temporary tables).

Standards

ANSI/ISO SQL Standard

CREATE TABLE is a Core Feature, though some of its components supported in the software are optional SQL Language Features. A subset of these features include:

- Temporary table support is SQL Language Feature F531.
- Support for IDENTITY columns is SQL Feature T174, though the software uses slightly different syntax from that in the standard.
- Foreign key constraint support includes SQL Language Features T191 "Referential action: RESTRICT", F741 "Referential MATCH types", F191 "Referential delete actions", and F701 "Referential update actions". The software does not support MATCH PARTIAL.
  The software does not support SQL Language Feature T591 ("UNIQUE constraints of possibly null columns"). In the software, all columns that are part of a PRIMARY KEY or UNIQUE constraint must be declared NOT NULL.

The following components of CREATE TABLE are not in the standard:

- The { IN | ON } `dbspace-name` clause.
- The ENCRYPTED, NOT TRANSACTIONAL, and SHARE BY ALL clauses.
- The COMPRESSED, INLINE, PREFIX, and NO INDEX clauses of a column definition.
- Various implementation-defined DEFAULT values, including AUTOINCREMENT, GLOBAL AUTOINCREMENT, CURRENT DATABASE, CURRENT REMOTE USER, CURRENT UTC TIMESTAMP, and most special values. A DEFAULT clause that references a SEQUENCE generator is also not in the standard.
- The specification of MATCH UNIQUE.
- Sortedness specification (ASC or DESC) on a PRIMARY KEY or FOREIGN KEY clause.
- The ability to specify FOREIGN KEY columns in an order different from that specified in the referenced table's PRIMARY KEY clause.

Example

The following statement creates a table file_table with two columns: file_name and file_contents. The contents column is LONG BINARY and is compressed:

```
CREATE TABLE file_table (
    file_name VARCHAR(255),
    file_contents LONG BINARY COMPRESSED
);
```

The following example creates a table for a library database to hold book information. A sample row is inserted into the table. Note that NOT NULL is assumed for primary key columns.

```
CREATE TABLE library_books (
    isbn CHAR(20) PRIMARY KEY,
    title CHAR(100) NOT NULL,
    author CHAR(50),
    copyright_date DATE
);
INSERT INTO library_books
   VALUES( '0-02-042710-7', 'The White Mountains', 'John Christopher', '1967');
```

The following example creates a table for a library database to hold information about borrowed books. The default value for date_borrowed indicates that the book is borrowed on the day the entry is made. The

date_returned column is NULL until the book is returned. When date_returned is NOT NULL, the CHECK constraint is evaluated. The sample INSERT statement is executed when the book is borrowed. The UPDATE statement is executed when the book is returned.

```
CREATE TABLE borrowed_books (
    book_id CHAR(20) REFERENCES library_books (isbn),
    borrower_id CHAR(15) NOT NULL,
    date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
    date_returned DATE,
    CHECK( date_returned >= date_borrowed )
);
INSERT INTO borrowed_books (book_id, borrower_id) values ( '0-02-042710-7',
'5198836300' );
UPDATE borrowed_books SET date_returned = CURRENT DATE
    WHERE book = '0-02-042710-7' AND borrower_id = '5198836300';
```

The following example creates tables for a sales database to hold order and order item information:

```
CREATE TABLE orders (
    order_num INTEGER NOT NULL PRIMARY KEY,
    date_ordered DATE,
    name CHAR(80)
);
CREATE TABLE order_items (
    order_num INTEGER NOT NULL,
    item_num SMALLINT NOT NULL,
    PRIMARY KEY ( order_num, item_num ),
    FOREIGN KEY ( order_num )
    REFERENCES orders ( order_num )
    ON DELETE CASCADE
);
```

The following example creates two tables named Table1 and Table2, adds foreign keys to Table2, and inserts values into Table1. The final statement attempts to insert values into Table2. An error is returned because the values that you attempt to insert are not a simple match with Table1.

```
CREATE TABLE Table1 ( P1 INT, P2 INT, P3 INT, P4 INT, P5 INT, P6 INT, PRIMARY
KEY ( P1, P2 ) );
CREATE TABLE Table2 ( F1 INT, F2 INT, F3 INT, PRIMARY KEY ( F1, F2 ) );
ALTER TABLE Table2
   ADD FOREIGN KEY fk2( F1, F2 )
   REFERENCES Table1( P1, P2 )
   MATCH SIMPLE;
INSERT INTO Table1 (P1, P2, P3, P4, P5, P6) VALUES ( 1, 2, 3, 4, 5, 6 );
INSERT INTO Table2 (F1, F2) VALUES ( 3, 4 );
```

The following statements show how MATCH SIMPLE and MATCH UNIQUE SIMPLE differ in the way multi-column foreign keys are handled when some but not all of the columns in the key are NULL. The final INSERT statement will fail because the values for the second column are not unique.

```
CREATE TABLE pt ( pk INT PRIMARY KEY, str VARCHAR(10) );
INSERT INTO pt VALUES (1,'one'), (2,'two');
COMMIT;
CREATE TABLE ft1 ( fpk INT PRIMARY KEY, FOREIGN KEY (ref) REFERENCES pt MATCH
SIMPLE );
INSERT INTO ft1 VALUES (100,1), (200,1);
CREATE TABLE ft2 ( fpk INT PRIMARY KEY, FOREIGN KEY (ref) REFERENCES pt MATCH
UNIQUE SIMPLE );
INSERT INTO ft2 VALUES (100,1), (200,1);
```

The following statements show how MATCH SIMPLE and MATCH FULL differ:

```
CREATE TABLE pt2 (
    pk1 INT NOT NULL,
    pk2 INT NOT NULL,
    str VARCHAR(10),
    PRIMARY KEY (pk1, pk2) );
INSERT INTO pt2 VALUES (1,10,'one-ten'), (2,20,'two-twenty');
COMMIT;
CREATE TABLE ft3 (
    fpk INT PRIMARY KEY,
    ref1 INT,
    ref2 INT );
ALTER TABLE ft3 ADD FOREIGN KEY (ref1, ref2)
    REFERENCES pt2 (pk1, pk2) MATCH SIMPLE;

CREATE TABLE ft4 (
    fpk INT PRIMARY KEY,
    ref1 INT,
    ref2 INT );
ALTER TABLE ft4 add FOREIGN KEY (ref1, ref2)
    REFERENCES pt2 (pk1, pk2) MATCH FULL;
/* 1 */ INSERT INTO ft3 VALUES (100, 1, 10);
/* 2 */ INSERT INTO ft3 VALUES (200, null, null);
/* 3 */ INSERT INTO ft3 VALUES (300, 2, null);
/* 4 */ INSERT INTO ft4 VALUES (100, 1, 10);
/* 5 */ INSERT INTO ft4 VALUES (200, null, null);
/* 6 */ INSERT INTO ft4 VALUES (300, 2, null);
```

The following list summarizes the result for each numbered INSERT statement above.

1. MATCH SIMPLE test succeeds; all column values match the corresponding values in pt2.
2. MATCH SIMPLE test succeeds; at least one column in the key is null.
3. MATCH SIMPLE test succeeds; at least one column in the key is null.
4. MATCH FULL test succeeds; all column values match the corresponding values in pt2.
5. MATCH FULL test succeeds; all column values in the key are null.
6. MATCH FULL test fails; both columns in the key must be null or match the corresponding values in pt2.

The second statement in the following example creates a table myT2 and sets the data type of its columns to the same data type of the corresponding columns in myT1 using the %TYPE attribute. Since additional attributes such as nullability are not applied, myT2.last_name will not have the same NOT NULL restriction that myT1.last_name has.

```
CREATE TABLE myT1 (
    first_name   CHAR(20),
    last_name    VARCHAR(30) NOT NULL
);
CREATE TABLE myT2 (
    first_name myT1.first_name%TYPE
    last_name myT1.last_name%TYPE
);
```

The following example creates two tables that will be used to demonstrate some of the features of the LIKE clause.

```
CREATE TABLE table1 ( id INT NOT NULL DEFAULT AUTOINCREMENT PRIMARY KEY, name
LONG VARCHAR );
CREATE TABLE table2 ( address LONG VARCHAR );
```

The following statement creates a new table just like table1:

```
CREATE TABLE table3
    LIKE table1
    INCLUDING IDENTITY
    INCLUDING PRIMARY KEY;
```

The following statement creates a new table like table1, but with additional columns from table2, and a column for a telephone number:

```
CREATE TABLE table4 (
    LIKE table1 INCLUDING ALL,
    LIKE table2,
    phone_number LONG VARCHAR
);
```

The following statement creates a table with all the data that is in the Departments table and the phone number of the head of each department from the Employees table. Since WITH DATA is the default, it need not be specified.

```
CREATE TABLE DepartmentsWithPhone
    AS ( SELECT D.*, E.Phone AS DepartmentPhone
        FROM Departments D JOIN Employees E
        ON D.DepartmentHeadID = E.EmployeeID )
    WITH DATA;
```

The following examples illustrate the creation of proxy tables that map to tables on remote servers.

The following example creates a table named t1 on the remote server toledo and creates a proxy table named proxy_t1 that is mapped to the remote table:

```
CREATE TABLE proxy_t1 ( a INT, b CHAR(10) )
AT 'toledo.db1.joe.t1';
```

The following example creates a proxy table named proxy_t2 that is mapped to the remote table t2 which already exists at the remote server toledo:

```
CREATE EXISTING TABLE proxy_t2 ( a INT, b CHAR(10) )
AT 'toledo.db1.joe.t2';
```

The following example creates a table named t3 with the attributes derived from the SELECT statement on the remote server toledo and creates a proxy table named proxy_t3 that is mapped to the remote table:

```
CREATE TABLE proxy_t3 AS (SELECT * FROM sa_rowgenerator(1,5))
AT 'toledo.db1.joe.t3';
```

The following example creates an encrypted table named t4 with the attributes derived from the SELECT statement but using different column names on the remote server toledo and creates a proxy table named proxy_t4 that is mapped to the remote table:

```
CREATE TABLE proxy_t4 (ident, firstname, lastname)
AS (SELECT ID,GivenName,Surname FROM Customers) ENCRYPTED
AT 'toledo.db1.joe.t4';
```

To utilize the ESCAPE CHARACTER clause, consider the following example:

1. Create two SQL Anywhere databases named test1.db and test2.db.

2. Start both databases on the same server:

```
dbsrv17 -n escape_test test1.db test2.db
```

3. Connect to test2 and create the following table:

```
CREATE TABLE "table.with;fun{characters}" (c int);
INSERT INTO "table.with;fun{characters}" VALUES(100);
COMMIT;
```

4. Disconnect and connect to test1.
5. Create a remote server to test2 as follows:

```
CREATE SERVER test2_server CLASS 'saodbc' USING 'driver=SQL Anywhere
Native;eng=escape_test;dbn=test2';
CREATE EXTERNLOGIN localuser TO test2_server REMOTE LOGIN remoteuser
IDENTIFIED BY remotepwd;
```

> **i Note**
>
> localuser is the userid used to log in to test1 while remoteuser and remotepwd are the remote userid and password needed to log in to test2.

The ESCAPE CHARACTER clause can be used to create a proxy table for the remote "table.with;fun{characters}" as follows:

```
CREATE EXISTING TABLE remtab AT 'test2_server;;;table.with!;fun!
{characters!}'ESCAPE CHARACTER'!';
```

OR

```
CREATE EXISTING TABLE remtab AT 'test2_server...table!.with!;fun!
{characters!}'ESCAPE CHARACTER'!';
```

Optionally, you can execute a query on the proxy table to ensure you get the expected result set back:

```
SELECT c FROM remtab;
```

The following examples illustrate the creation of temporary tables.

The following statement creates a simple global temporary table:

```
CREATE GLOBAL TEMPORARY TABLE customers0 (ID int, firstname char(30), lastname
char(30));
```

The following statement creates an empty global temporary table with the same columns and attributes as the Customers table:

```
CREATE GLOBAL TEMPORARY TABLE customers1
LIKE Customers INCLUDING ALL;
```

The following statement creates a populated global temporary table with some of the columns and data of the Customers table but with different column names :

```
CREATE GLOBAL TEMPORARY TABLE customers2 (ident, firstname, lastname)
AS (SELECT ID, GivenName, Surname FROM Customers);
```

The following statement creates an empty shared global temporary table with some of the columns of the Customers table but with different column names :

```
CREATE GLOBAL TEMPORARY TABLE customers3 (ident, firstname, lastname)
AS (SELECT ID, GivenName, Surname FROM Customers)
ON COMMIT PRESERVE ROWS
SHARE BY ALL
WITH NO DATA;
```

The following statement creates a populated local temporary table with some of the columns of the Customers table but with different column names :

```
CREATE LOCAL TEMPORARY TABLE customers3 (ident, firstname, lastname)
AS (SELECT ID, GivenName, Surname FROM Customers)
ON COMMIT PRESERVE ROWS
WITH DATA;
```

Related Information

SQL Variables [page 123]
Strings [page 11]
OLAP Support
Reloading Tables with AUTOINCREMENT Columns
The Special IDENTITY Column
Special Values [page 87]
Temporary Tables
Computed Columns
Additional Dbspaces Considerations
Use of a Sequence to Generate Unique Values
%TYPE and %ROWTYPE Attributes [page 115]
Encrypting a Table
Creating a Table
@@identity Global Variable [page 126]
ALTER TABLE Statement [page 742]
CREATE EXISTING TABLE Statement [page 854]
DECLARE LOCAL TEMPORARY TABLE Statement [page 1066]
SQL Data Types [page 129]
CREATE DBSPACE Statement [page 830]

1.4.4.97 CREATE TEMPORARY TRACE EVENT Statement

Creates a user trace event that persists until the database is stopped.

Syntax

```
CREATE [ OR REPLACE ] TEMPORARY TRACE EVENT trace-event-name
```

```
[ WITH SEVERITY {
 0-255
 | ALWAYS
 | CRITICAL
 | ERROR
 | WARNING
 | INFORMATION
 | DEBUG } ]
[ ( field-name field-type [ , ... ] ] )
```

Parameters

trace-event-name

Specify a user trace event name. User trace event names cannot start with the prefix SYS_. System trace event names cannot be specified.

OR REPLACE clause

Create a new trace event, or replace an existing trace event with the same name.

WITH SEVERITY clause

If the severity level is not specified, the default severity level (DEBUG) is used. User trace events are owned by the database that the user was connected to when the trace event was created. The supported severity values are:

Level	Severity value range
ALWAYS	0
CRITICAL	1-50
ERROR	51-100
WARNING	101-150
INFORMATION	151-200
DEBUG	201-255

field-name

The field that gathers information of a specific type from the user trace event. The field must be a valid identifier.

field-type

You can use any data type that is supported for a column except an array type.

Remarks

A trace event is stored in memory and is dropped when the database server stops if it has not been dropped explicitly.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a user trace event:

```
CREATE TEMPORARY TRACE EVENT my_event( id INTEGER, information LONG VARCHAR );
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.4.4.98 CREATE TEMPORARY TRACE EVENT SESSION Statement

Creates a user trace event session.

Syntax

```
CREATE [ OR REPLACE ] TEMPORARY TRACE EVENT SESSION session-name
[ ON SERVER ]
event-definition [ ,... ]
[ target-definition ]
```

```
event-definition :
ADD TRACE EVENT event-name [ ( WHERE search-condition ) ]
```

```
target-definition :
ADD TARGET FILE
  ( SET target-parameter-name=target-parameter-value [ ,... ] )
```

```
target-parameter-name :
{ filename_prefix
  | max_size
  | num_files
  | flush_on_write
  | ["compressed"] }
```

Parameters

OR REPLACE clause

Specifying CREATE OR REPLACE creates a trace event session or replaces an existing trace event session with the same name.

ON SERVER clause

The trace event session records trace events from all databases on the database server. If this clause is not specified, then the trace event session only records trace events from the database on which the session is created.

WHERE clause

The WHERE clause allows an event to be traced conditionally based on its properties, and can contain expressions that refer to constants and event fields. If there are built-in functions used in `search-condition`, then they are evaluated on the connection generating the event. For example, using `connection_property('number') = 101` logs only events for connections with the number 101.

`search-condition` is applied before events are sent to `target-definition`. If `search-condition` returns FALSE (or UNKNOWN), then the event is not sent to the target.

`search-condition` returns the error if it contains any of the following:

- Sub-queries

- User-defined functions
- Sequences
- Host variables
- Column references (may refer to fields of the event but not WHERE clause of the event)
- Connection or database variables

session-name

The name of the trace event session.

event-name

The name of the trace event to add to the session. System- and user-defined trace events are supported. Call the sp_trace_events system procedure to obtain a list of system-defined trace events.

target-name

The only supported value is FILE.

target-parameter-name

The following target parameters are supported:

target-parameter-name	target-parameter-value
filename_prefix	An ETD file name prefix with or without a path. ETD files have the extension `.etd`. This parameter is required.
max_size	The maximum size of the file in bytes. The default is 0, which means there is no limit on the file size and it grows as long as disk space is available. Once the specified size is reached, a new file is started.
num_files	Use this option to limit the number of files when max_size is set to a non-zero value. It is the number of additional files to preserve when event tracing information is split over many files. The default is 0, which means there is no limit on the number of files. When a file reaches its maximum size, the database server starts writing a new file. Each file has a file name suffix _N where N starts at 0. When num_files is not 0, older files are automatically removed. For example, if num_files is 2 and the current file number suffix is _100, then files with suffix _100, _99, and _98 are kept (the current file and 2 others). The file with suffix _97 is deleted. As each new file is written, this number suffix increases by 1.
flush_on_write	A boolean (true or false) value that controls whether disk buffers are flushed for each event that is logged. The default is false. When flushing is enabled, the performance of the database server may be reduced if many trace events are being logged.
[compressed]	A boolean (true or false) value that controls compression of the ETD file to conserve disk space. The default is false. Use brackets with this parameter name because it is a keyword in other contexts.

Remarks

Trace event sessions do not run until they are explicitly started with the ALTER TRACE EVENT SESSION statement. Trace event sessions capture trace events related to system behavior or for a particular user. Trace event sessions are stored in memory and are dropped when the database server stops if they have not been dropped explicitly.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates an event tracing session that records information about the user-defined event my_event and the system-defined event SYS_ConsoleLog_Information to a file named `my_trace_file`:

```
CREATE TEMPORARY TRACE EVENT SESSION my_session
    ADD TRACE EVENT my_event, -- user event
    ADD TRACE EVENT SYS_ConsoleLog_Information -- system event
    ADD TARGET FILE (SET filename_prefix='my_trace_file', [compressed]=1); --
add a target
```

The following statement creates an event tracing session called MySession that records connection events for user DBA:

```
CREATE TEMPORARY TRACE EVENT SESSION MySession
ADD TRACE EVENT SYS_RLL_Connect
( WHERE user='DBA' );
```

Related Information

Event Tracing
System Events
Search Conditions [page 58]
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)
-sf Database Server Option

1.4.4.99 CREATE TEXT CONFIGURATION Statement

Creates a text configuration object for use with building and updating text indexes.

Syntax

```
CREATE TEXT CONFIGURATION [ owner.]new-config-name
   FROM [ owner.]existing-config-name
```

Parameters

FROM clause

Specify the name of a text configuration object to use as the template for creating the new one. The names of the default text configuration objects are default_char and default_nchar.

When you create a text configuration object, the database options that affect how date and time values are converted to strings are copied from the default_char and default_nchar text configuration object templates.

Remarks

You create a text configuration object using another text configuration object as a template and then alter the options as needed using the ALTER TEXT CONFIGURATION statement.

To view the list of all text configuration objects in the database, and their settings, query the SYSTEXTCONFIG system view.

Privileges

You must have the CREATE TEXT CONFIGURATION system privilege to create text configurations objects owned by you. You must have the CREATE ANY TEXT CONFIGURATION or CREATE ANY OBJECT system privilege to create text configuration objects owned by others.

All text configuration objects have PUBLIC access. Any user with privilege to create a text index can also use any text configuration object.

Side Effects

Automatic commit

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following CREATE TEXT CONFIGURATION statement creates a text configuration object, max_term_sixteen, using the default_char text configuration object. The subsequent ALTER TEXT CONFIGURATION statement changes the maximum term length for max_term_sixteen to 16.

```
CREATE TEXT CONFIGURATION max_term_sixteen FROM default_char;
ALTER TEXT CONFIGURATION max_term_sixteen
   MAXIMUM TERM LENGTH 16;
```

Related Information

Full Text Search
Example Text Configuration Objects
Database Options That Impact Text Configuration Objects
What to Specify When Creating or Altering Text Configuration Objects
Tutorial: Performing a Full Text Search on a GENERIC TeIt index
Tutorial: Performing a Fuzzy Full Text Search
SYSTEXTCONFIG System View [page 1958]
ALTER TEXT CONFIGURATION Statement [page 757]
DROP TEXT CONFIGURATION Statement [page 1135]
sa_refresh_text_indexes System Procedure [page 1683]

1.4.4.100 CREATE TEXT INDEX Statement

Creates a text index.

Syntax

```
CREATE TEXT INDEX [ IF NOT EXISTS ] text-index-name
ON [ owner. ] { table-name | mv-name } ( column-name, ... )
  [ IN dbspace-name ]
  [ CONFIGURATION [ owner. ]text-configuration-name ]
  [ { IMMEDIATE REFRESH
    | MANUAL REFRESH
    | AUTO REFRESH [ EVERY integer { MINUTES | HOURS } ] } ]
```

Parameters

IF NOT EXISTS clause

When the IF NOT EXISTS clause is specified and the named text index exists, no changes are made and an error is not returned.

ON clause

Specify the table and columns on which to build the text index.

IN clause

Specify the dbspace in which the text index is located. If this clause is not specified, then the text index is created in the same dbspace as the table it references.

CONFIGURATION clause

Specify the text configuration object to use when creating the text index. If this clause is not specified, the default_nchar text configuration object is used if any of the columns in the index are NCHAR; otherwise, the default_char text configuration object is used.

REFRESH clause

Specify the refresh type for the text index. If you do not specify a REFRESH clause, IMMEDIATE REFRESH is used as the default. You can specify the following refresh types:

IMMEDIATE REFRESH

Refreshes the text index each time changes in the underlying table or the materialized view impact data in the text index.

AUTO REFRESH

Refreshes the text index automatically using an internal server event. Use the EVERY sub-clause to specify the refresh interval in minutes or hours. If you specify AUTO REFRESH without supplying interval information, the database server refreshes the text index every 60 minutes. A text index may be refreshed earlier than the interval specified by the AUTO REFRESH clause if the pending_size value, as returned by the sa_text_index_stats system procedure, exceeds 20% of the text index size at the last refresh or if the deleted_length exceeds 50% of the text index size. An internal event executes once per minute to check this condition for all AUTO REFRESH text indexes.

MANUAL REFRESH

The text index is refreshed manually.

Remarks

Creating more than one text index referencing a column can return unexpected results, and is not recommended.

You cannot create a text index on a regular view or a temporary table.

Once a text index is created on a materialized view, it cannot be refreshed or truncated, it can only be dropped. The text index on a materialized view is maintained by the database server whenever the underlying materialized view is refreshed or updated.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

An IMMEDIATE REFRESH text index on a base table is populated at creation time and an exclusive lock is held on the table during this initial refresh. IMMEDIATE REFRESH text indexes provide full support for queries that use snapshot isolation.

An IMMEDIATE REFRESH text index on a materialized view is populated at creation time if the view is populated.

MANUAL and AUTO REFRESH text indexes must be initialized (refreshed) after creation.

Refreshes for AUTO REFRESH text indexes scan the table using isolation level 0.

Once a text index is created, you cannot change it to, or from, being defined as IMMEDIATE REFRESH. If either of these changes is required, drop and recreate the text index.

You can choose to manually refresh an AUTO REFRESH text index by using the REFRESH TEXT INDEX statement.

Privileges

To create a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

To create a text index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

Side Effects

Automatic commit

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a text index, myTxtIdx, on the Description column of the MarketingInformation table in the sample database. The MarketingTextConfig text configuration object is used, and the refresh interval is set to every 24 hours.

```
CREATE TEXT INDEX myTxtIdx ON GROUPO.MarketingInformation ( Description )
   CONFIGURATION default_char
   AUTO REFRESH EVERY 24 HOURS;
```

Related Information

Snapshot Isolation
Text Index Refresh Types
Text Index Concepts and Reference
Tutorial: Performing a Full Text Search on a GENERIC TeIt index
Viewing Text Index Terms and Settings (SQL Central)

Tutorial: Performing a Fuzzy Full Text Search
isolation_level Option
REFRESH TEXT INDEX Statement [page 1325]
SYSTEXTCONFIG System View [page 1958]
ALTER TEXT INDEX Statement [page 762]
DROP TEXT INDEX Statement [page 1136]
REFRESH TEXT INDEX Statement [page 1325]
TRUNCATE TEXT INDEX Statement [page 1445]
sa_char_terms System Procedure [page 1534]
sa_nchar_terms System Procedure [page 1664]
sa_refresh_text_indexes System Procedure [page 1683]
sa_text_index_stats System Procedure [page 1735]

1.4.4.101 CREATE TIME ZONE Statement

Creates a time zone object that can be used by the database to simulate a time zone that is different from the server's time zone.

Syntax

```
CREATE [ OR REPLACE ] TIME ZONE
name time-zone-option [ ... ]
```

```
time-zone-option :
 { OFFSET offset
 | DST OFFSET offset
 | NO DST
 | [ DST ] STARTING month-day-rule AT { minutes | hours:minutes }
 | [ DST ] ENDING month-day-rule AT { minutes | hours:minutes } }
```

```
offset :
'{ + | -} { minutes | hours:minutes } '
```

```
month-day-rule :
'month / day-rule'
```

```
day-rule :
number
| LAST day
| day >= number
```

Parameters

OR REPLACE clause

Creates a new time zone or replaces an existing time zone with the same name. If the time zone already exists and OR REPLACE is not specified, then the statement fails.

OFFSET clause

Specifies the standard offset for the time zone in hours and minutes ahead of UTC. If the offset is a whole number, then it indicates minutes, so 1:00 and 60 are equivalent. The offset cannot be more than +23:59 (1439 minutes) or less than -23:59 (-1439 minutes).

DST OFFSET clause

Specifies the difference from standard time that results from using daylight savings time. The default is NO DST if none of the DST clauses are used. If the STARTING and ENDING clauses are specified but the DST OFFSET clause is not, then the default is one hour.

NO DST clause

Specifies that the time zone does not observe daylight savings time. You cannot use the STARTING, ENDING, or DST OFFSET clauses with this clause.

[DST] STARTING clause

Specifies the date and time when daylight savings time begins.

[DST] ENDING clause

Specifies the date and time when daylight savings time ends.

month-day-rule

`month` is a number from 1 to 12 or an English three-letter short form (Jan, Feb, and so on).

day-rule

The day of the month on which to start or end daylight savings time.

number

A number from 1 to 31. If specifying `number`, then `day-rule` is a date within the month. For example, specifying **`Mar/12`** for `month-day-rule` indicates March 12.

last day

The last specified week day of the month. For example, specifying **`last Fri`**, indicates the last Friday of the month.

day

A number from 1 to 7 or an English three-letter short form (Mon, Tue, and so on).

day >= number

The earliest day either on or after a particular date of the month. `number` must be less than or equal to the number of days in the month. For example, **`Feb/Mon>=28`**.

> **i** Note
>
> `day` **`>=`** `number` may match a date in a different month. For example, the DST rule 'Sep / Wed >= 25' (the first Wednesday after September 25), might be in October, depending on the year.

Remarks

The DST clauses are optional. If the time zone observes daylight savings time, then the STARTING and ENDING clauses must be specified.

Multiple simulated time zone objects can be created for a database. If you have started two or more databases, each database can run in its own simulated time zone that is different from the server's actual time zone. Use the SET OPTION PUBLIC.time_zone statement to switch to the desired time zone.

Privileges

You must have the MANAGE TIME ZONE system privilege to create time zones.

Side Effects

Automatic commit.

Executing this statement populates the ISYSTIMEZONE system table.

Example

To add the Eastern Time zone, execute the following statement:

```
CREATE TIME ZONE EST5EDT
OFFSET '-05:00'
DST STARTING 'Mar/Sun>=8' AT '2:00'
DST ENDING 'Nov/Sun>=1' AT '2:00';
```

The time zone is five hours behind UTC. Daylight savings time starts at 2:00 A.M. on the first Sunday on or after March 8 (the second Sunday of March), and ends at 2:00 A.M. on the first Sunday of November.

To add the Australian Eastern Time zone, execute the following statement:

```
CREATE TIME ZONE NewSouthWales
OFFSET '10:00'
DST STARTING 'Oct/Sun>=1' AT '2:00'
DST ENDING 'Apr/Sun>=1' AT '2:00';
```

The time zone is ten hours ahead of UTC. Daylight savings time begins on the first Sunday of October at 2:00 A.M., and ends on the first Sunday of April at 2:00 A.M.

To switch to the NewSouthWales time zone, execute the following statement:

```
SET OPTION PUBLIC.time_zone='NewSouthWales';
```

To return to the server's time zone, execute the following statement:

```
SET OPTION PUBLIC.time_zone=;
```

Related Information

COMMENT Statement [page 808]
ALTER TIME ZONE Statement [page 764]
DROP TIME ZONE Statement [page 1138]
time_zone Option
SYSTIMEZONE System View [page 1961]

1.4.4.102 CREATE TRIGGER Statement

Creates a trigger on a table.

Syntax

```
CREATE [ OR REPLACE ] TRIGGER [owner.][table-name.]trigger-name trigger-type
{ trigger-event-list | UPDATE OF column-list }
[ ORDER integer ] ON table-name
[ REFERENCING [ OLD AS old-name ]
   [ NEW AS new-name ]
   [ REMOTE AS remote-name ] ]
[ FOR EACH { ROW | STATEMENT } ]
[ WHEN ( search-condition ) ]
trigger-body
```

```
column-list :  column-name[, ...]
```

```
trigger-type :
BEFORE
| AFTER
| INSTEAD OF
| RESOLVE
```

```
trigger-event-list : trigger-event[, ...  ]
```

```
trigger-event :
DELETE
| INSERT
| UPDATE
```

```
trigger-body : a BEGIN statement that optionally includes boolean logic
keywords ({ IF | ELSIF } { INSERTING | UPDATING | DELETING } THEN some-action)
```

Parameters

OR REPLACE clause

Specifying OR REPLACE creates a new trigger, or replaces an existing trigger with the same name.

trigger-type

Row-level triggers can be defined to execute BEFORE, AFTER, or INSTEAD OF an insert, update, or delete operation. Statement-level triggers can be defined to execute INSTEAD OF or AFTER the statement.

BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whenever the new value differs from the old value. That is, if a `column-list` is specified for a BEFORE UPDATE trigger, then the trigger fires if any of the columns in `column-list` appear in the SET clause of the UPDATE statement. If a `column-list` is specified for an AFTER UPDATE trigger, then the trigger is fired only if the value of any of the columns in `column-list` is *changed* by the UPDATE statement.

INSTEAD OF triggers are the only form of trigger that you can define on a regular view. INSTEAD OF triggers replace the triggering action with another action. When an INSTEAD OF trigger fires, the triggering action is skipped and the specified action is performed. INSTEAD OF triggers can be defined as a row-level or a statement-level trigger. A statement-level INSTEAD OF trigger replaces the entire statement, including all row-level operations. If a statement-level INSTEAD OF trigger fires, then no row-level triggers fire as a result of that statement. However, the body of the statement-level trigger could perform other operations that, in turn, cause other row-level triggers to fire.

If you are defining an INSTEAD OF trigger, then you cannot use the UPDATE OF `column-list` clause, the ORDER clause, or the WHEN clause.

The RESOLVE trigger type is for use with SQL Remote; it fires before row-level UPDATE or UPDATE OF `column-list` only.

trigger-event

When defining a trigger, you can combine DELETE, INSERT, and UPDATE events in the same definition, but triggers for UPDATE OF events must be defined separately. You can define any number of DELETE, INSERT, and UPDATE triggers on a table. You can define any number of triggers for UPDATE OF events on a table, but only one per column.

Triggers can be fired by the following events:

DELETE event

The trigger is invoked whenever one or more rows of the table are deleted.

INSERT event

The trigger is invoked whenever one or more rows are inserted into the table.

UPDATE event

The trigger is invoked whenever one or more rows of the table are updated.

The keyword UPDATING is also supported for this clause for compatibility with other SQL dialects. The argument for UPDATING is a quoted string (for example, `UPDATING( 'mycolumn' )`), whereas the argument for UPDATE is an identifier (for example, `UPDATE( mycolumn )`).

UPDATE OF column-list event

The trigger is invoked whenever a row of the associated table is updated and a column in the `column-list` is modified. This type of trigger event cannot be used in a `trigger-event-list`; it must be the only trigger event defined for the trigger. This clause cannot be used in an INSTEAD OF trigger.

You can only specify one UPDATE OF trigger per column.

You can write separate triggers for each event that you need to handle or, if you have some shared actions and some actions that depend on the event, you can create a trigger for all events and use an IF statement to distinguish the action taking place.

ORDER clause

It is good practice to specify order for triggers when defining multiple triggers on a table, even if they are not the same type. This ensures predictable results and makes easier to confirm which order they are processed in.

When defining additional triggers of the same type (insert, update, or delete) to fire at the same time (before, after, or resolve), you must specify an ORDER clause to tell the database server the order in which to fire the triggers. Order numbers must be unique among same-type triggers configured to fire at the same time. If you specify an order number that is not unique, then an error is returned. Order numbers do not need to be in consecutive order (for example, you could specify 1, 12, 30). The database server fires the triggers starting with the lowest number.

Typically, if you omit the ORDER clause, or specify 0, then the database server assigns the order of 1. However, if another same-type trigger is already set to 1, then an error is returned.

When you create additional triggers that contain *multiple* event types, if you omit the ORDER clause, and one or more of the event types is the same as in other triggers (for example, the *trigger-event-list* for one trigger is UPDATE, INSERT, and the *trigger-event-list* for another trigger is UPDATE), the database server does not return an error. In this case, the database server processes the triggers in an implementation-specific order that may not be expected and is subject to change. Therefore, it is strongly recommended that you always specify an ORDER clause when defining more than one trigger on a table.

When adding additional triggers, you may need to modify the existing same-type triggers for the event, depending on whether the actions of the triggers interact. If they do not interact, then the new trigger must have an ORDER value unique from other existing triggers. If they do interact, you need to consider what the other triggers do, and you may need to change the order in which they fire.

The ORDER clause is not supported for INSTEAD OF triggers since there can only be one INSTEAD OF trigger of each type (insert, update, or delete) defined on a table or view.

REFERENCING clause

The REFERENCING OLD and REFERENCING NEW clauses allow you to refer to the inserted, deleted, or updated rows. With this clause an UPDATE is treated as a delete followed by an insert.

An INSERT takes the REFERENCING NEW clause, which represents the inserted row. There is no REFERENCING OLD clause.

A DELETE takes the REFERENCING OLD clause, which represents the deleted row. There is no REFERENCING NEW clause.

An UPDATE takes the REFERENCING OLD clause, which represents the row before the update, and it takes the REFERENCING NEW clause, which represents the row after the update.

The meanings of REFERENCING OLD and REFERENCING NEW differ, depending on whether the trigger is a row-level or a statement-level trigger. For row-level triggers, the REFERENCING OLD clause allows you to refer to the values in a row before an update or delete, and the REFERENCING NEW clause allows you to

refer to the inserted or updated values. The OLD and NEW rows can be referenced in BEFORE and AFTER triggers. The REFERENCING NEW clause allows you to modify the new row in a BEFORE trigger before the insert or update operation takes place.

For statement-level triggers, the REFERENCING OLD and REFERENCING NEW clauses refer to declared temporary tables holding the old and new values of the rows.

The REFERENCING REMOTE clause is for use with SQL Remote. It allows you to refer to the values in the VERIFY clause of an UPDATE statement. It should be used only with RESOLVE UPDATE or RESOLVE UPDATE OF column-list triggers.

FOR EACH clause

To declare a trigger as a row-level trigger, use the FOR EACH ROW clause. To declare a trigger as a statement-level trigger, you can either use a FOR EACH STATEMENT clause or omit the FOR EACH clause. For clarity, it is recommended that you specify the FOR EACH STATEMENT clause if you are declaring a statement-level trigger.

WHEN clause

The trigger fires only for rows where the search-condition evaluates to true. The WHEN clause can be used only with row level triggers. This clause cannot be used in an INSTEAD OF trigger.

trigger-body

The trigger body contains the actions to take when the triggering action occurs, and consists of a BEGIN statement.

You can include trigger operation conditions in the BEGIN statement. Trigger operation conditions perform actions depending on the trigger event that caused the trigger to fire. For example, if the trigger is defined to fire for both updates and deletes, you can specify different actions for the two conditions.

You can also use Boolean conditions `{ INSERTING | DELETING | UPDATING [ ( 'col-name' ) ] }` anywhere a condition can be used in the body of the trigger. This special syntax enables you to specify an additional action to take when performing some `trigger-event`. For example, `IF INSERTING THEN SET msg = msg || 'insert'`.

Remarks

The CREATE TRIGGER statement creates a trigger associated with a table in the database, and stores the trigger in the database.

You cannot define a trigger on a materialized view. If you do, a SQLE_INVALID_TRIGGER_MATVIEW error is returned.

A trigger is declared as either a row-level trigger, in which case it executes before or after each row is modified, or a statement-level trigger, in which case it executes after the entire triggering statement is completed.

CREATE TRIGGER puts a table lock on the table and requires exclusive use of the table.

Privileges

You must have the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege. Additionally, you must be the owner of the table the trigger is built on or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege

To create a trigger on a view owned by someone else, you must have either the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege, and you must have either the ALTER ANY VIEW or ALTER ANY OBJECT system privilege.

To replace an existing trigger, you must be the owner of the table the trigger is built on, or have one of the following:

- CREATE ANY TRIGGER system privilege.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY TRIGGER system privileges.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

CREATE TRIGGER is part of optional ANSI/ISO SQL Language Feature T211 "Basic trigger capability". Row triggers are optional ANSI/ISO SQL Language Feature T212, while INSTEAD OF triggers are optional ANSI/ISO SQL Language Feature T213.

Some trigger features in the software are not in the standard. These include:

- The optional OR REPLACE syntax. If an existing trigger is replaced, authorization of the creation of the new trigger instance is bypassed.
- The ORDER clause. In the ANSI/ISO SQL Standard, triggers are fired in the order they were created.
- RESOLVE triggers.

Transact-SQL

ROW and RESOLVE triggers are not supported by Adaptive Server Enterprise. The Transact-SQL dialect does not support Transact-SQL INSTEAD OF triggers, though these are supported by Adaptive Server Enterprise. Transact-SQL triggers are defined using different syntax.

Example

This example creates a statement-level trigger. First, create a table as shown in this CREATE TABLE statement (requires the CREATE TABLE system privilege):

```
CREATE TABLE t0
( id INTEGER NOT NULL,
 times TIMESTAMP NULL DEFAULT CURRENT TIMESTAMP,
 remarks TEXT NULL,
 PRIMARY KEY ( id )
);
```

Next, create a statement-level trigger for this table:

```
CREATE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0
REFERENCING NEW AS new_name
FOR EACH STATEMENT
BEGIN
  DECLARE @id1 INTEGER;
  DECLARE @times1 TIMESTAMP;
  DECLARE @remarks1 LONG VARCHAR;
  DECLARE @err_notfound EXCEPTION FOR SQLSTATE VALUE '02000';
//declare a cursor for table new_name
  DECLARE new1 CURSOR FOR
   SELECT id, times, remarks FROM new_name;
  OPEN new1;
 //Open the cursor, and get the value
  LoopGetRow:
  LOOP
      FETCH NEXT new1 INTO @id1, @times1,@remarks1;
      IF SQLSTATE = @err_notfound THEN
   LEAVE LoopGetRow
      END IF;
      //print the value or for other use
      PRINT (@remarks1);
  END LOOP LoopGetRow;
  CLOSE new1
END;
```

The following example replaces the myTrig trigger created in the previous example.

```
CREATE OR REPLACE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0
REFERENCING NEW AS new_name
FOR EACH STATEMENT
BEGIN
FOR L1 AS new1 CURSOR FOR
   SELECT id, times, remarks FROM new_name
DO
     //print the value or for other use
     PRINT (@remarks1);
END FOR;
END;
```

The next example shows how you can use REFERENCING NEW in a BEFORE UPDATE trigger. This example ensures that postal codes in the new Employees table are in uppercase. You must have the SELECT, ALTER, and UPDATE object-level privileges on GROUPO.Employees to execute this statement:

```
CREATE TRIGGER emp_upper_postal_code
BEFORE UPDATE OF PostalCode
ON GROUPO.Employees
REFERENCING NEW AS new_emp
FOR EACH ROW
WHEN ( ISNUMERIC( new_emp.PostalCode ) = 0 )
```

```
BEGIN
   -- Ensure postal code is uppercase (employee might be
   -- in Canada where postal codes contain letters)
   SET new_emp.PostalCode = UPPER(new_emp.PostalCode)
END;
UPDATE GROUPO.Employees SET state='ON', PostalCode='n2x 4y7' WHERE
EmployeeID=191;
SELECT PostalCode FROM GROUPO.Employees WHERE EmployeeID = 191;
```

The next example shows how you can use REFERENCING OLD in a BEFORE DELETE trigger. This example prevents deleting an employee from the Employees table who has not been terminated.

```
CREATE TRIGGER TR_check_delete_employee
BEFORE DELETE
ON Employees
REFERENCING OLD AS current_employee
FOR EACH ROW WHEN ( current_employee.Terminate IS NULL )
BEGIN
    RAISERROR 30001 'You cannot delete an employee who has not been fired';
END;
```

The next example shows how you can use REFERENCING NEW and REFERENCING OLD in a BEFORE UPDATE trigger. This example prevents a decrease in an employee's salary.

```
CREATE TRIGGER TR_check_salary_decrease
    BEFORE UPDATE
      ON GROUPO.Employees
    REFERENCING OLD AS before_update
    NEW AS after_update
FOR EACH ROW
BEGIN
   IF after_update.salary < before_update.salary THEN
    RAISERROR 30002 'You cannot decrease a salary';
    END IF;
END;
```

The next example shows how you can use REFERENCING NEW in a BEFORE INSERT and UPDATE trigger. The following example creates a trigger that fires before a row in the SalesOrderItems table is inserted or updated.

```
CREATE TRIGGER TR_update_date
   BEFORE INSERT, UPDATE
     ON GROUPO.SalesOrderItems
   REFERENCING NEW AS new_row
FOR EACH ROW
BEGIN
   SET new_row.ShipDate = CURRENT TIMESTAMP;
END;
```

The following trigger displays a message on the *History* tab of the Interactive SQL *Results* pane showing which action caused the trigger to fire.

```
CREATE TRIGGER tr BEFORE INSERT, UPDATE, DELETE
ON sample_table
REFERENCING OLD AS t1old
FOR EACH ROW
BEGIN
    DECLARE msg varchar(255);
    SET msg = 'This trigger was fired by an ';
    IF INSERTING THEN
        SET msg = msg || 'insert'
    ELSEIF DELETING THEN
        set msg = msg || 'delete'
    ELSEIF UPDATING THEN
```

```
        set msg = msg || 'update'
    END IF;
    MESSAGE msg TO CLIENT
END;
```

Related Information

INSTEAD OF Triggers
SQL Statements for Implementing Integrity Constraints
Triggers
Stored Procedures, Triggers, Batches, and User-defined Functions
Creating a Trigger on a Table (SQL Central)
BEGIN Statement [page 784]
CREATE TRIGGER Statement [T-SQL] [page 1043]
DROP TRIGGER Statement [page 1142]
ROLLBACK TRIGGER Statement [page 1358]
UPDATE Statement [page 1463]
ALTER TRIGGER Statement [page 768]
RAISERROR Statement [page 1315]
CONFLICT Function [Miscellaneous] [page 288]

1.4.4.103 CREATE TRIGGER Statement [T-SQL]

Creates a new trigger in the database in a manner compatible with Adaptive Server Enterprise.

Syntax

General use

```
CREATE TRIGGER [owner.]trigger_name
ON [owner.]table_name
FOR { INSERT, UPDATE, DELETE }
AS statement-list
```

Specifying additional conditions

```
CREATE TRIGGER [owner.]trigger_name
ON [owner.]table_name
FOR {INSERT, UPDATE}
AS
[ IF UPDATE ( column-name )
[ { AND | OR } UPDATE ( column-name ) ] ... ]
  statement-list
[ IF UPDATE ( column-name )
[ { AND | OR} UPDATE ( column-name ) ] ... ]
  statement-list
```

Remarks

CREATE TRIGGER acquires an exclusive table lock on the table.

The rows deleted or inserted are held in two temporary tables. In the Transact-SQL form of triggers, they can be accessed using the table names "deleted", and "inserted", as in Adaptive Server Enterprise. In the Watcom SQL CREATE TRIGGER statement, these rows are referenced using the REFERENCING clause.

Trigger names must be unique in the database.

Transact-SQL triggers are executed AFTER the triggering statement has executed.

Since the ORDER clause is not supported when creating Transact-SQL triggers, the value of `trigger_order` is set to 1. The SYSTRIGGER system table has a unique index on: `table_id`, `event`, `trigger_time`, and `trigger_order`. For a particular event (insert, update, delete), statement-level triggers are always AFTER and `trigger_order` cannot be set, so there can be only one of each type per table, assuming any other triggers do not set an order other than 1.

Privileges

You must be the owner of the table, or have ALTER privilege on the table, or have ALTER ANY TABLE system privilege. Additionally, you must have the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

ROW triggers are not supported by Adaptive Server Enterprise. The SQL Anywhere Transact-SQL dialect does not support Transact-SQL INSTEAD OF triggers, though these are supported by Adaptive Server Enterprise.

If an `owner` is specified for `trigger_name`, it is ignored. SQL Anywhere triggers do not have owners.

Related Information

Transact-SQL Triggers

CREATE TRIGGER Statement [page 1036]

1.4.4.104 CREATE USER Statement

Creates a database user or group.

Syntax

```
CREATE USER user-name [ IDENTIFIED BY password ]
  [ LOGIN POLICY policy-name ]
  [ FORCE PASSWORD CHANGE { ON | OFF } ]
```

Parameters

user-name

The name of the user you are creating. The user name is an identifier which must follow the rules for an identifier.

password

The password of the user. You can specify an identifier or a string literal as the password. If you specify an identifier then you must follow the rules for an identifier. If you specify a string literal, then there are fewer restrictions.

policy-name

The name of the login policy to assign the user. If no login policy is specified, the DEFAULT login policy is applied.

FORCE PASSWORD CHANGE clause

Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the user's login policy.

Remarks

You do not have to specify an IDENTIFIED BY clause for the user. A user with no password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID.

The verify_password_function option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement.

Subject to the restrictions of minimum password length, a user can be created with an empty password, in which case they do not need to specify a password when logging in. This is not the same as creating a user with no password. Creating such a user is not advisable.

```
CREATE USER userid IDENTIFIED BY "";
```

You can use the ALTER USER statement to change your own password.

If you use the CREATE USER statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY USER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following examples create a user named SQLTester, each with the same password Welcome. All of these examples specify a valid identifier for the password.

```
CREATE USER SQLTester IDENTIFIED BY Welcome;
CREATE USER SQLTester IDENTIFIED BY "Welcome";
CREATE USER SQLTester IDENTIFIED BY `Welcome`;
CREATE USER SQLTester IDENTIFIED BY [Welcome];
```

The following example creates a user named SQLTester with the password Wel[come]. This example specifies a string literal for the password which permits the use of square brackets.

```
CREATE USER SQLTester IDENTIFIED BY 'Wel[come]';
```

The following example creates a user named SQLTester with the password Welcome. The SQLTester user will be required to specify a new password upon logging in for the first time.

```
CREATE USER SQLTester IDENTIFIED BY Welcome
FORCE PASSWORD CHANGE ON;
```

Related Information

Password and User ID Restrictions and Considerations
Login Policies
Creating a User (SQL Central)
CREATE LOGIN POLICY Statement [page 894]
ALTER LOGIN POLICY Statement [page 690]
ALTER USER Statement [page 769]
COMMENT Statement [page 808]
DROP LOGIN POLICY Statement [page 1101]
DROP USER Statement [page 1143]
GRANT Statement [page 1193]
GRANT ROLE Statement [page 1210]
min_password_length Option

1.4.4.105 CREATE VARIABLE Statement

Creates a connection- or database-scope variable.

Syntax

Create a connection-scope variable

```
CREATE [ OR REPLACE ] VARIABLE identifier data-type [ { = | DEFAULT } initial-value ]
```

```
initial-value : expression
```

Create a database-scope variable

```
CREATE [ OR REPLACE ] DATABASE VARIABLE [ IF NOT EXISTS ] [ owner.]identifier data-type [ { = | DEFAULT } initial-value ]
```

```
initial-value : expression
```

Parameters

OR REPLACE clause

Specifying the OR REPLACE clause drops the named variable if it already exists and recreates it with the new definition. OR REPLACE only replaces the value of the variable if the data type of the current and new value are the same.

Do not use this clause with the IF NOT EXISTS clause.

owner

This parameter applies only to database-scope variables. Specify a valid user ID or role, or PUBLIC to set ownership of the variable. If set to a user, only that user can use the database variable. If set to a role, users who have that role are able to use the database variable. If set to PUBLIC, all users are able to use the variable.

If `owner` is not specified, it is set to the user executing the CREATE VARIABLE statement.

identifier

A valid identifier for the variable.

data-type

The data type for the variable. Set the data type explicitly, or use the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a variable or a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a cursor, table, or view.

%ROWTYPE and TABLE REF is not supported as data types for database-scope variables.

IF NOT EXISTS clause

Specify this clause to allow the statement to complete without returning an error if a database-scope variable with the same name already exists. This parameter is only for use when creating owned database-scope variables.

Do not use this clause with the OR REPLACE clause.

{ = | DEFAULT } clause

The default value for the variable. For database-scope variables, this is also the initial value after the database is restarted.

`initial-value` must match the data type defined by `data-type`. If you do not specify an `initial-value`, then the variable contains the NULL value until a different value is assigned, for example by using a SET statement, a SELECT ... INTO statement, or in an UPDATE statement. If `initial-value` is set by using an expression, then the expression is evaluated at creation time and the resulting constant is stored (not the expression).

Remarks

Variables are useful for sharing values between procedures. They are also useful for creating large text or binary objects for INSERT or UPDATE statements from Embedded SQL programs. Database-scope variables are useful for sharing values across connections and database restarts.

Use the CREATE VARIABLE syntax to create a connection-scope variable that is available in the context of the connection.

Use the CREATE DATABASE VARIABLE syntax to create a database-scope variable that can be used by other users and other connections.

Use the OR REPLACE clause as an alternative to the VAREXISTS function in SQL scripts.

If you specify a variable name for `initial-value`, then the variable must already be initialized in the database.

Privileges

Connection-scope variables: No privileges are required to create or replace a connection-scope variable.

Database-scope variables: To create or replace a self-owned database-scope variable, you must have the CREATE DATABASE VARIABLE or MANAGE ANY DATABASE VARIABLE system privilege. To create or replace a database-scope variable owned by another user or by PUBLIC, you must have the MANAGE ANY DATABASE VARIABLE system privilege.

Side Effects

Connection-scope variables: There are no side effects associated with creating a connection-scope variable.

Database-scope variables: Creating and replacing a database-scope variable causes an automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example creates (or updates) a database-scope variable called site_name of type VARCHAR(50).

```
CREATE OR REPLACE DATABASE VARIABLE @site_name VARCHAR(50);
```

This example creates (or updates) a database-scope variable owned by PUBLIC called database_name of type CHAR(66) and sets it to the special value CURRENT DATABASE.

```
CREATE OR REPLACE DATABASE VARIABLE PUBLIC.@database_name CHAR(66) DEFAULT
CURRENT DATABASE;
```

This example creates a connection-scope variable named first_name, of data type VARCHAR(50).

```
CREATE VARIABLE first_name VARCHAR(50);
```

This example creates a connection-scope variable named birthday, of data type DATE.

```
CREATE VARIABLE birthday DATE;
```

This example creates a connection-scope variable named v1 as an INT with the initial setting of 5.

```
CREATE VARIABLE v1 INT = 5;
```

This example creates a connection-scope variable named v1 and sets its value to 10, regardless of whether the v1 variable already exists.

```
CREATE OR REPLACE VARIABLE v1 INT = 10;
```

This example creates a connection-scope variable, ProductID, and uses the %TYPE attribute to set its data type to the data type of the ID column in the Products table:

```
CREATE VARIABLE ProductID Products.ID%TYPE;
```

This example creates a connection-scope variable, ItemsForSale, and uses the %ROWTYPE attribute to set its data type to a composite data type comprised of the columns defined for the Products table. It then creates another variable, ItemID, and declares its type to be the data type of the ID column in the ItemsForSale variable:

```
CREATE VARIABLE ItemsForSale Products%ROWTYPE;
CREATE VARIABLE ItemID ItemsForSale.ID%TYPE;
```

Related Information

SQL Variables [page 123]
Compound Statements
Special Values [page 87]
%TYPE and %ROWTYPE Attributes [page 115]
BEGIN Statement [page 784]
SQL Data Types [page 129]
DROP VARIABLE Statement [page 1145]
SET Statement [page 1373]
VAREXISTS Function [Miscellaneous] [page 620]
SYSDATABASEVARIABLE System View [page 1969]

1.4.4.106 CREATE VIEW Statement

Creates a view on the database.

Syntax

```
CREATE [ OR REPLACE ] VIEW
[ owner.]view-name [ ( column-name, ... ) ]
AS query-expression
[ WITH CHECK OPTION ]
```

Parameters

OR REPLACE clause

Specifying OR REPLACE (CREATE OR REPLACE VIEW) creates a view or replaces an existing view with the same name. Existing privileges are preserved when you use the OR REPLACE clause, but INSTEAD OF triggers on the view are dropped.

If you execute a CREATE OR REPLACE VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. Drop the trigger before altering or dropping the view.

AS clause

The SELECT statement on which the view is based. The SELECT statement must not refer to local temporary tables. Also, `query-expression` can have a GROUP BY, HAVING, WINDOW, or ORDER BY clause, and can contain UNION, EXCEPT, INTERSECT, or a common table expression.

Query semantics dictate that the order of the rows returned is undefined unless the query combines an ORDER BY clause with a TOP or FIRST clause in the SELECT statement.

WITH CHECK OPTION clause

The WITH CHECK OPTION clause rejects any updates and inserts to the view that do not meet the criteria of the view as defined by its `query-expression`.

Remarks

Views do not physically exist in the database as tables. They are derived each time they are used. A view is derived as the result of a SELECT statement specified in a CREATE VIEW statement. In a view, specifying the user ID of the table owner is recommended to distinguish tables with the same name.

A view name can be used in place of a table name in SELECT, DELETE, UPDATE, and INSERT statements.

SELECT * can be used in the main query, a subquery, a derived table, or a subselect of the CREATE VIEW statement.

A query can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine

the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

Views can be updated unless the `query-expression` defining the view contains a GROUP BY clause, a WINDOW clause, an aggregate function, or involves a set operator (UNION, INTERSECT, EXCEPT). An update to the view updates the underlying table(s).

The view's columns are given the names specified in the `column-name` list. If the column name list is not specified, view columns are given names from the SELECT list items. All items in the SELECT list must have unique names. To use names from the SELECT list items, each item must be a simple column name or have a specified alias.

The database server does permit unnamed expressions in the SELECT list of the `query-expression` referenced in the CREATE VIEW statement. Unnamed expressions in the SELECT list of the `query-expression` are assigned the name **expression**, concatenated with an integer value if more than one such expression exists. For example, the following statement would define view V with three columns (expression, expression1, and expression2), and these names would appear in the SYSCOLUMN system view for the created view V.

```
CREATE VIEW V AS
   SELECT DATEADD( DAY, 1, NOW() ), DATEADD( DAY, 2, NOW() ), DATEADD( DAY, 2,
NOW() )
   FROM SYS.DUMMY;
```

Relying on these generated names is not recommended since other views with unnamed SELECT list expressions have the identical assigned names.

Typically, a view references tables and views (and their respective attributes) that are defined in the catalog. However, a view can also reference SQL variables (with the exception of TABLE REF variables). When variables are used in the definition, when a query that references the view is executed, the value of the SQL variable is substituted for the variable. Views that reference SQL variables are called **parameterized views** since the variables act as parameters to the execution of the view. Parameterized views offer an alternative to embedding the body of an equivalent SELECT block in a query as a derived table in the query's FROM clause. Parameterized views can also be useful for queries that are embedded in stored procedures where the SQL variables referenced in the view are input parameters for the procedure.

It is not necessary for the SQL variable to exist when the CREATE VIEW statement is executed. However, if the SQL variable is not defined when a query that refers to the view is executed, an error is returned indicating that the column (variable) could not be found.

Variables of type TABLE REF (table reference variables) are not permitted in the outermost SELECT list of a view definition.

Privileges

You must have the CREATE VIEW system privilege to create views owned by you. You must have the CREATE ANY VIEW or CREATE ANY OBJECT system privilege to create views owned by others.

To replace an existing view, you must be the owner of the view, or have one of the following:

- CREATE ANY VIEW and DROP ANY VIEW system privileges.

- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY VIEW system privileges.

If the tables referenced by the view are owned by other users, you must have the SELECT object-level privileges on those tables.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Core Feature, but some features of a view's embedded SELECT statement are optional Language Features. The ability to specify an ORDER BY clause with the top-level SELECT statement in the view definition is optional Language Feature F852. Restricting the result set of a view using SELECT TOP or LIMIT is optional Language Feature F859. Specifying WITH CHECK OPTION on a view that is not updatable, for example, the view's SELECT statement contains a derived table involving aggregation or DISTINCT, or a set operator (INTERSECT, EXCEPT, or UNION), is optional Language Feature T111.

The following extensions are also not in the ANSI/ISO SQL Standard: parameterized views, the optional OR REPLACE syntax, and the automatic generation of names for unnamed SELECT list expressions.

Example

The following example creates a view showing information for male employees only. This view has the same column names as the base table:

```
CREATE VIEW MaleEmployees
AS SELECT *
FROM GROUPO.Employees
WHERE Sex = 'M';
```

The following example creates a view showing employees and the departments they belong to:

```
CREATE VIEW EmployeesAndDepartments
  AS SELECT Surname, GivenName, DepartmentName
  FROM GROUPO.Employees JOIN GROUPO.Departments
  ON Employees.DepartmentID = Departments.DepartmentID;
```

The following example replaces the EmployeesAndDepartments view created in the previous example. After replacing the view, the view shows the city, state, and country location for each employee:

```
CREATE OR REPLACE VIEW EmployeesAndDepartments
  AS SELECT Surname, GivenName, City, State, Country
  FROM GROUPO.Employees JOIN GROUPO.Departments
```

```
  ON Employees.DepartmentID = Departments.DepartmentID;
```

The following example creates a parameterized view based on the variables var1 and var2, which are not attributes of the Employees or Departments tables:

```
CREATE VIEW EmployeesByState
  AS SELECT Surname, GivenName, DepartmentName
  FROM GROUPO.Employees JOIN GROUPO.Departments
  ON Employees.DepartmentID = Departments.DepartmentID
  WHERE Employees.State = var1 and Employees.Status = var2;
```

Variables can appear in the view's SELECT statement in any context where a variable is a permitted expression. For example, the following parameterized view utilizes the parameter var1 as the pattern for a LIKE predicate:

```
CREATE VIEW ProductsByDescription
  AS SELECT *
  FROM GROUPO.Products
  WHERE Products.Description LIKE var1;
```

To use this view, define the variable var1 before executing the query that references the view. For example, the following BEGIN statement could be placed in a procedure, function, or a batch statement:

```
BEGIN
DECLARE var1 CHAR(20);
SET var1 = '%cap%';
SELECT * FROM ProductsByDescription
END
```

Related Information

INSTEAD OF Triggers
Creating a Regular View
ALTER VIEW Statement [page 772]
SELECT Statement [page 1362]

1.4.4.107 DEALLOCATE DESCRIPTOR Statement [ESQL]

Frees memory associated with a SQL descriptor area.

Syntax

```
DEALLOCATE DESCRIPTOR descriptor-name
```

```
descriptor-name : identifier
```

Remarks

Frees all memory associated with a descriptor area, including the data items, indicator variables, and the structure itself.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

DEALLOCATE DESCRIPTOR is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL".

Related Information

The SQL Descriptor Area (SQLDA)
ALLOCATE DESCRIPTOR Statement [ESQL] [page 660]
SET DESCRIPTOR Statement [ESQL] [page 1380]

1.4.4.108 DEALLOCATE Statement

This statement has no effect in SQL Anywhere, and is ignored. It is provided for compatibility with Adaptive Server Enterprise and Microsoft SQL Server. Refer to your Adaptive Server Enterprise or Microsoft SQL Server documentation for more information about this statement.

Standards

ANSI/ISO SQL Standard

Not in the standard.

1.4.4.109 Declaration Section [ESQL]

Declares host variables in an Embedded SQL program. Host variables are used to exchange data with the database.

Syntax

```
EXEC SQL BEGIN DECLARE SECTION;
C declarations
EXEC SQL END DECLARE SECTION;
```

Remarks

A declaration section is simply a section of C variable declarations surrounded by the BEGIN DECLARE SECTION and END DECLARE SECTION statements. A declaration section makes the SQL preprocessor aware of C variables that are used as host variables. Not all C declarations are valid inside a declaration section.

Privileges

None.

Standards

ANSI/ISO SQL Standard

Core Feature.

Example

```
EXEC SQL BEGIN DECLARE SECTION;
char *surname, initials[5];
int dept;
EXEC SQL END DECLARE SECTION;
```

Related Information

Host Variables in Embedded SQL
BEGIN Statement [page 784]

1.4.4.110 DECLARE Statement

Declares a SQL variable (connection-scope) or an exception within a compound statement (BEGIN...END).

Syntax

Declaring a variable

```
DECLARE identifier [, ... ] data-type
[ { = | DEFAULT } initial-value ]
```

```
initial-value : expression
```

Declaring an exception

```
DECLARE exception-name EXCEPTION
FOR SQLSTATE [ VALUE ] string
```

Parameters

identifier

A valid identifier for the variable.

data-type

The data type for the variable. Set the data type explicitly, or you can set it by using the %TYPE or %ROWTYPE attribute. Use %TYPE to set it to the data type of a variable or a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a cursor, table, or view.

DEFAULT clause

The default value for the variable. If you specify `initial-value`, the data type must match the type defined by `data-type`.

initial-value

The default and initial value for the variable. `initial-value` must match the data type defined by `data-type`. If you do not specify an `initial-value`, the variable contains the NULL value until a different value is assigned.

Remarks

DECLARE `identifier`: Variables used in the body of a procedure, trigger, or batch can be declared using the DECLARE statement. The variable persists for the duration of the compound statement in which it is declared.

If you specify a variable name for `initial-value`, the variable must already be initialized in the database.

The body of a Watcom SQL procedure or trigger is a compound statement, and variables must be declared with other declarations, such as a cursor declaration (DECLARE CURSOR), immediately following the BEGIN keyword. In a Transact-SQL procedure or trigger, there is no such restriction.

DECLARE `exception-name` **EXCEPTION:** Use this syntax to declare variables for SQL language exceptions within a compound statement (BEGIN...END). The variables can be used, for example, for comparison with the SQLSTATEs obtained during execution, with the SIGNAL statement, or as part of the exception case within the exception handler.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Declaring variables syntax - Persistent Stored Module feature. **Declaring exceptions syntax** - The form of exception declaration supported by the software, namely the DECLARE EXCEPTION statement, is not in the standard; in the ANSI/ISO SQL Standard, exceptions are specified using a handler declaration using the keywords DECLARE HANDLER. The DECLARE...EXCEPTION syntax is not allowed in T-SQL procedures.

Transact-SQL

The syntax for declaring exceptions cannot be used in Transact-SQL compound statements and procedures.

Example

The following batch illustrates the use of the DECLARE statement and prints a message in the database server messages window:

```
BEGIN
    DECLARE varname CHAR(61);
    SET varname = 'Test name';
    MESSAGE varname;
END;
```

The following example declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```
BEGIN
    DECLARE v1 INT = 5;
    DECLARE v2, v3 CHAR(10) = 'abc';
END;
```

The following example declares two variables and sets their values using host parameters:

```
BEGIN
    DECLARE p1, p2 VARCHAR(32);
    SELECT ?, ? INTO p1, p2;
END;
```

The following procedure declares an exception for use with the SQLSTATE comparison:

```
CREATE PROCEDURE HighSales( IN cutoff INT, OUT HighValues INT )
BEGIN
    DECLARE err_notfound EXCEPTION FOR
        SQLSTATE '02000';
    DECLARE curThisCust CURSOR FOR
        SELECT CAST( sum( SalesOrderItems.Quantity *
            Products.UnitPrice ) AS INTEGER) VALUE
        FROM Customers
            LEFT OUTER JOIN SalesOrders
            LEFT OUTER JOIN SalesOrderItems
            LEFT OUTER JOIN Products
        GROUP BY CompanyName;
    DECLARE ThisValue INT;
    SET HighValues = 0;
    OPEN curThisCust;
    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust INTO ThisValue;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop;
        END IF;
        IF ThisValue > cutoff THEN
            SET HighValues = HighValues + ThisValue;
        END IF;
    END LOOP CustomerLoop;
    CLOSE curThisCust;
END;
```

The following compound statement declares an exception for use with SIGNAL and an exception handler:

```
BEGIN
    DECLARE err_div_by_0 EXCEPTION FOR
```

```
        SQLSTATE '22012';
    DECLARE curQuantity CURSOR FOR
        SELECT Quantity
        FROM SalesOrderItems
        WHERE ProductID = 300;
    DECLARE Quantities INT;
    DECLARE altogether INT;
    SET Quantities = 0;
    SET altogether = 0;
    OPEN curQuantity;
    LOOP
        FETCH NEXT curQuantity
            INTO Quantities;
        IF SQLSTATE = '02000' THEN
            SIGNAL err_div_by_0;
        END IF;
        SET altogether = altogether + Quantities;
    END LOOP;
EXCEPTION
    WHEN err_div_by_0 THEN
        CLOSE curQuantity;
        SELECT altogether;
        RETURN;
    WHEN OTHERS THEN
       RESIGNAL;
END;
```

The following example creates a procedure, DepartmentsCloseToCustomerLocation, and sets its IN parameter to the data type of the ID column in the Customers table by using a %TYPE attribute:

```
CREATE OR REPLACE PROCEDURE DepartmentsCloseToCustomerLocation( IN customer_ID
Customers.ID%TYPE )
BEGIN
    DECLARE cust_rec Customers%ROWTYPE;
    SELECT City, State, Country
    INTO cust_rec.City, cust_rec.State, cust_rec.Country
    FROM Customers
    WHERE ID = customer_ID;
    SELECT Employees.Surname, Employees.GivenName, Departments.DepartmentName
    FROM Employees JOIN Departments
    ON Departments.DepartmentHeadID = Employees.EmployeeID
    WHERE Employees.City = cust_rec.City
    AND Employees.State = cust_rec.State
    AND Employees.Country = cust_rec.Country;
END;
CALL DepartmentsCloseToCustomerLocation(158);
```

The following example uses the %ROWTYPE attribute to create a variable, @a_product, and then inserts data from the Products table into the variable:

```
CREATE OR REPLACE PROCEDURE CheckStock( IN @id Products.ID%TYPE )
BEGIN
    DECLARE @a_product Products%ROWTYPE;
    SET (@a_product).ID = 200;
END;
```

Related Information

Special Values [page 87]
Exception Handlers

%TYPE and %ROWTYPE Attributes [page 115]
SQL Data Types [page 129]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
BEGIN Statement [page 784]
SQLSTATE Special Value [page 109]

1.4.4.111 DECLARE CURSOR Statement [ESQL] [SP]

Declares a cursor.

Syntax

Embedded SQL

```
DECLARE cursor-name
[ UNIQUE ]
[ NO SCROLL
  | DYNAMIC SCROLL
  | SCROLL
  | INSENSITIVE
  | SENSITIVE ]
CURSOR FOR
{ select-statement
| statement-name
| call-statement }
```

Stored procedures

```
DECLARE cursor-name
[ NO SCROLL
  | DYNAMIC SCROLL
  | SCROLL
  | INSENSITIVE
  | SENSITIVE ]
CURSOR
{ FOR select-statement
| FOR call-statement
| USING variable-name }
```

```
cursor-name : identifier
```

```
statement-name : identifier | hostvar
```

```
variable-name : identifier
```

Parameters

UNIQUE clause

When a cursor is declared UNIQUE, the query is forced to return all the columns required to uniquely identify each row. Often this means ensuring that all columns in the primary key or a uniqueness table

constraint are returned. Any columns that are required but were not specified in the query are added to the result set.

A DESCRIBE done on a UNIQUE cursor sets the following additional options in the indicator variables:

DT_KEY_COLUMN

The column is part of the key for the row.

DT_HIDDEN_COLUMN

The column was added to the query because it was required to uniquely identify the rows.

NO SCROLL clause

A cursor declared NO SCROLL is restricted to moving forward through the result set using FETCH NEXT and FETCH RELATIVE 0 seek operations.

As rows cannot be returned to once the cursor leaves the row, there are no sensitivity restrictions on the cursor. When a NO SCROLL cursor is requested, the database server supplies the most efficient kind of cursor, which is an asensitive cursor.

DYNAMIC SCROLL clause

DYNAMIC SCROLL is the default cursor type. DYNAMIC SCROLL cursors can use all formats of the FETCH statement.

When a DYNAMIC SCROLL cursor is requested, the database server supplies an asensitive cursor. When using cursors there is always a trade-off between efficiency and consistency. Asensitive cursors provide efficient performance at the expense of consistency.

SCROLL clause

A cursor declared SCROLL can use all formats of the FETCH statement. When a SCROLL cursor is requested, the database server supplies a value-sensitive cursor. With a value-sensitive cursor, a subsequent fetch of a previously fetched result row may return a warning or an error if the underlying row has been modified or deleted.

The database server must execute value-sensitive cursors in such a way that result set membership is guaranteed. DYNAMIC SCROLL cursors are more efficient and should be used unless the consistent behavior of SCROLL cursors is required.

INSENSITIVE clause

A cursor declared INSENSITIVE has its membership fixed when it is opened; a temporary table is created with a copy of all the original rows. FETCHING from an INSENSITIVE cursor does not see the effect of any other INSERT, UPDATE, or DELETE statement from concurrently executing transactions, or any other update operations from within the same transaction. INSENSITIVE cursors are not updatable.

SENSITIVE clause

A cursor declared SENSITIVE is sensitive to changes to membership or values of the result set.

FOR statement-name clause

Statements are named using the PREPARE statement. Cursors can be declared only for a prepared SELECT or CALL. The cursor updatability specified in the PREPARE statement is used for the cursor, unless the SQL preprocessor -m HISTORICAL option is specified.

USING variable-name clause

For use within stored procedures only. The variable is a string containing a SELECT statement for the cursor. The variable must be available when the DECLARE is processed, and so must be either a parameter to the procedure, or nested inside another BEGIN...END after the variable has been assigned a value.

Remarks

Cursors are the primary means for manipulating the results of queries. The DECLARE CURSOR statement declares a cursor with the specified name for a SELECT statement or a CALL statement. In a Watcom SQL procedure, trigger, or batch, a DECLARE CURSOR statement must appear with other declarations, immediately following the BEGIN keyword. Cursor names must be unique.

If a cursor is declared inside a compound statement, it exists only for the duration of that compound statement (whether it is declared in a Watcom SQL or Transact-SQL compound statement).

When a single statement is processed, all of the DECLARE CURSOR statements must use distinct names, even if the cursors are declared in scopes that do not overlap. However, the cursor can only be used within the compound statement that declares it.

The type of cursor specified in a DECLARE CURSOR statement can dictate the execution plan selected by the query optimizer for that statement. For example, an INSENSITIVE cursor over a SELECT statement requires the complete materialization of the result set of the SELECT statement when the cursor is opened. Moreover, the type of cursor must match the characteristics of the underlying statement. If there is a mismatch between the cursor type and the statement, then the cursor type may be changed automatically. For example, an INSENSITIVE cursor declaration conflicts with an updatable SELECT statement that specifies FOR UPDATE, since by definition INSENSITIVE cursors are read only. In this case, the cursor type is changed automatically from INSENSITIVE to an updatable, value-sensitive cursor when the cursor is opened.

If the updatability of a SELECT statement embedded in a cursor declaration is unspecified, it is determined by the setting of the ansi_update_constraints option.

Privileges

None.

Side Effects

If the cursor type must be changed to satisfy the requirements of the underlying statement, a warning is returned when the cursor is opened.

Standards

ANSI/ISO SQL Standard

DECLARE CURSOR is a Core Feature. The ability to specify FOR UPDATE with SCROLL or NO SCROLL is optional ANSI/ISO SQL Language Feature F831, "Full cursor update". Using DECLARE CURSOR in an

Embedded SQL program constitutes optional ANSI/ISO SQL Language Feature B031. Some cursor types are also optional ANSI/ISO SQL features. These include:

- INSENSITIVE cursors are optional SQL language feature F791 of the ANSI/ISO SQL Standard.
- SENSITIVE cursors are optional SQL language feature F231 of the ANSI/ISO SQL Standard.
- Scrollable cursors are optional SQL language feature F431 of the ANSI/ISO SQL Standard.

The software supports a number of extensions to DECLARE CURSOR, as follows:

- The software supports several extensions to the FOR UPDATE clause, which the ANSI/ISO SQL Standard defines as a clause of the DECLARE CURSOR statement.
- WITH HOLD is specified as a clause of the OPEN statement, rather than as a clause of the DECLARE CURSOR statement as defined in this ANSI/ISO SQL Standard.
- The ANSI/ISO SQL Standard separates the notions of cursor sensitivity and scrollability, while for historical reasons the software combines the two. In the software, all cursors are forward and backward scrollable except for those declared as NO SCROLL.
- DYNAMIC SCROLL and UNIQUE are not in the ANSI/ISO SQL Standard. DYNAMIC SCROLL has similar behavior to cursors declared as ASENSITIVE in the ANSI/ISO SQL Standard.
- The ability to declare a cursor over a CALL statement, or with a USING clause, is not in the ANSI/ISO SQL Standard.

Transact-SQL

DECLARE CURSOR is supported by Adaptive Server Enterprise, but there are several behavioral differences. Adaptive Server Enterprise differentiates, as in the ANSI/ISO SQL Standard, between scrollability and sensitivity; in Adaptive Server Enterprise, cursor sensitivity options are SEMI-SENSITIVE, INSENSITIVE, or default (akin to ASENSITIVE). In Adaptive Server Enterprise, NO SCROLL cursors are the default, and all scrollable cursors are read-only. Several features of the DECLARE CURSOR statement are not supported by Adaptive Server Enterprise. These include:

- Adaptive Server Enterprise does not support the SQL Anywhere cursor concurrency clause. To acquire a lock on a fetched row, you must use the HOLDLOCK table hint.
- Adaptive Server Enterprise does not support DYNAMIC SCROLL or UNIQUE cursors. DYNAMIC SCROLL is similar to Adaptive Server Enterprise default cursor behavior.
- The ability to declare a cursor over a CALL statement, or with a USING clause, is not supported by Adaptive Server Enterprise.

In Adaptive Server Enterprise, Transact-SQL procedures and functions can contain multiple DECLARE CURSOR statements that use the same cursor name. In Adaptive Server Enterprise, the DEALLOCATE CURSOR statement is used to eliminate a cursor from the current scope, so that a subsequent OPEN statement can reference the correct, previously declared cursor. This feature is not supported in SQL Anywhere. In SQL Anywhere, all cursors in a given scope must have unique names. If a Transact-SQL dialect procedure contains multiple cursor declarations with the same name, the procedure parses without error. However, at execution time, if a second DECLARE CURSOR statement with the same cursor name is executed, an error occurs.

You should be aware that the TDS wire protocol for Open Client and jConnect connections does not implement true scrollable result sets. When scrolling backward through a cursor, the FETCH request may be satisfied immediately if the desired row is within a window of prefetched rows that have already been retrieved by the TDS client. If the desired row is beyond this window, however, the cursor's SELECT statement may be re-executed.

Example

The following example illustrates how to declare a scroll cursor in Embedded SQL:

```
EXEC SQL DECLARE cur_employee SCROLL CURSOR
FOR SELECT * FROM GROUPO.Employees;
```

The following example illustrates how to declare a cursor for a prepared statement in Embedded SQL:

```
EXEC SQL PREPARE employee_statement
FROM 'SELECT Surname FROM GROUPO.Employees'FOR READ ONLY;
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement;
```

The following example illustrates the use of cursors in a stored procedure:

```
BEGIN
  DECLARE cur_employee CURSOR FOR
      SELECT Surname
      FROM GROUPO.Employees;
  DECLARE name CHAR(40);
  OPEN cur_employee;
  lp: LOOP
    FETCH NEXT cur_employee INTO name;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    ...
  END LOOP;
  CLOSE cur_employee;
END
```

This example shows the USING clause being used as a parameter to the procedure:

```
CREATE FUNCTION GetRowCount( IN qry LONG VARCHAR )
RETURNS INT
BEGIN
  DECLARE crsr CURSOR USING qry;
  DECLARE rowcnt INT;
  SET rowcnt = 0;
  OPEN crsr;
  lp: LOOP
    FETCH crsr;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    SET rowcnt = rowcnt + 1;
  END LOOP;
  CLOSE crsr;
  RETURN rowcnt;
END;
```

This example shows the USING clause nested inside a BEGIN...END, after `variable-name` has been assigned a value.

```
CREATE PROCEDURE get_table_name(
  IN id_value INT, OUT tabname CHAR(128)
)
BEGIN
  DECLARE qry LONG VARCHAR;
  SET qry = 'SELECT table_name FROM SYS.SYSTAB ' ||
            'WHERE table_id=' || string(id_value);
  BEGIN
    DECLARE crsr CURSOR USING qry;
    OPEN crsr;
    FETCH crsr INTO tabname;
```

```
    CLOSE crsr;
  END
END;
```

The following example returns an error as the two cursor names within the statement are not unique:

```
BEGIN
   BEGIN
     DECLARE MyCursor DYNAMIC SCROLL CURSOR FOR SELECT 1;
   END;
   BEGIN
     DECLARE MYCursor DYNAMIC SCROLL CURSOR FOR SELECT 2;
   END;
END;
```

The following example returns an error since the cursor has not been declared within the statement that is trying to open it.

```
BEGIN
   BEGIN
     DECLARE MyCursor DYNAMIC SCROLL CURSOR FOR SELECT 1;
   END;
   BEGIN
     OPEN MyCursor;
   END;
END;
```

Related Information

Sensitive Cursors
Insensitive Cursors
Value-sensitive Cursors
Asensitive Cursors
PREPARE Statement [ESQL] [page 1308]
OPEN Statement [ESQL] [SP] [page 1286]
EXPLAIN Statement [ESQL] [page 1160]
SELECT Statement [page 1362]
CALL Statement [page 795]
FOR Statement [page 1166]
The Embedded SQL Preprocessor
ansi_update_constraints Option

1.4.4.112 DECLARE LOCAL TEMPORARY TABLE Statement

Declares a local temporary table.

Syntax

```
DECLARE LOCAL TEMPORARY TABLE table-name
```

```
[ ( { column-definition | table-constraint | pct-free | like-clause }, ... ) |
as-clause | like-clause ]
[ ON COMMIT { DELETE | PRESERVE } ROWS
  | NOT TRANSACTIONAL ]
```

```
column-definition :
column-name data-type
[ COMPRESSED ]
[ INLINE { inline-length | USE DEFAULT } ]
[ PREFIX { prefix-length | USE DEFAULT } ]
[ [ NO ] INDEX ]
[ [ NOT ] NULL ]
[ DEFAULT default-value | IDENTITY | COMPUTE ( expression ) ]
[ column-constraint ... ]
[ table-element ... ]
```

```
default-value :
  special-value
| string
| global-variable
| [ - ] number
| ( constant-expression )
| ( sequence-expression )
| built-in-function( constant-expression )
| AUTOINCREMENT
```

```
special-value :
  CURRENT DATABASE
| CURRENT DATE
| CURRENT TIME
| [ CURRENT ] TIMESTAMP
| [ CURRENT ] USER
| [ CURRENT ] UTC TIMESTAMP
| EXECUTING USER
| INVOKING USER
| LAST USER
| NULL
| PROCEDURE OWNER
| SESSION USER
```

```
column-constraint :
  [ CONSTRAINT constraint-name ]
  { UNIQUE [ CLUSTERED | NONCLUSTERED ]
  | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ]
  | CHECK ( condition )
  }
```

```
table-element :
 { column-definition
 | table-constraint-definition
 | like-clause }
```

```
table-constraint :
  [ CONSTRAINT constraint-name ]
  {   UNIQUE [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC | DESC ], ... )
    | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC |
DESC ], ... )
    | CHECK ( condition )
  }
```

```
pct-free : PCTFREE percent-free-space

percent-free-space : integer

as-clause :
[ (column-name, ... ) ] AS ( select-statement )

like-clause :
LIKE source-table [ like-option [ ... ] ]

like-option :
{ INCLUDING | EXCLUDING } option

option :
  ALL
| IDENTITY
| DEFAULTS
| CONSTRAINTS
| INDEXES
| PRIMARY KEY
| FOREIGN KEYS
| COMMENTS
| STORAGE
```

Parameters

See the CREATE TABLE statement for a description of DECLARE LOCAL TEMPORARY TABLE clauses.

Remarks

The DECLARE LOCAL TEMPORARY TABLE statement declares a temporary table.

The DECLARE LOCAL TEMPORARY TABLE...LIKE syntax declares a new table based directly on the definitions of another table. You can also clone a table with additional columns, constraints, and LIKE clauses, or create a table based on a SELECT statement.

Tables created using DECLARE LOCAL TEMPORARY TABLE do not appear in the SYSTABLE view of the system catalog.

The rows of a declared temporary table are deleted when the table is explicitly dropped or when the table goes out of scope. You can also explicitly delete rows using TRUNCATE or DELETE.

Declared local temporary tables within compound statements exist within the compound statement. Otherwise, the declared local temporary table exists until the end of the connection.

Two local temporary tables within the same scope cannot have the same name. If you create temporary table with the same name as a base table, the base table only becomes visible within the connection once the scope of the local temporary table ends. A connection cannot create a base table with the same name as an existing temporary table.

If you want a procedure to create a local temporary table that persists after the procedure completes, use the CREATE LOCAL TEMPORARY TABLE statement instead.

You cannot use the REFERENCES `column-constraint` or the FOREIGN KEY table-constraint on a local temporary table.

The use of %TYPE or %ROWTYPE is not allowed for variable or temporary objects.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

DECLARE LOCAL TEMPORARY TABLE is part of optional Language Feature F531. The PCTFREE and NOT TRANSACTIONAL clauses are not in the standard. The column and constraint definitions defined by the statement may also include extension syntax that is not in the standard. In the ANSI/ISO SQL Standard, tables created via the DECLARE LOCAL TEMPORARY TABLE statement appear in the system catalog; however, this is not the case in the software.

Transact-SQL

DECLARE LOCAL TEMPORARY TABLE is not supported by Adaptive Server Enterprise. In SAP Adaptive Server Enterprise, one creates a temporary table using the CREATE TABLE statement with a table name that begins with the special character #.

Example

DECLARE statements must appear first after a BEGIN statement. The following example illustrates how to declare a temporary table in a stored procedure:

```
BEGIN
  DECLARE LOCAL TEMPORARY TABLE TempTab ( number INT );
  ...
END
```

The following example declares two local temporary tables that will be used to demonstrate some of the features of the LIKE clause.

```
DECLARE LOCAL TEMPORARY TABLE table1 ( id INT NOT NULL DEFAULT AUTOINCREMENT
PRIMARY KEY, name LONG VARCHAR );
DECLARE LOCAL TEMPORARY TABLE table2 ( address LONG VARCHAR );
```

The following statement declares a new local temporary table just like table1:

```
DECLARE LOCAL TEMPORARY TABLE table3
    LIKE table1
    INCLUDING IDENTITY
    INCLUDING PRIMARY KEY;
```

The following statement declares a new local temporary table like table1, but with additional columns from table2, and a column for a telephone number:

```
DECLARE LOCAL TEMPORARY TABLE table4 (
    LIKE table1 INCLUDING ALL,
    LIKE table2,
    phone_number LONG VARCHAR
);
```

The following statement creates a table with all the data that is in the Departments table and the phone number of the head of each department from the Employees table. Since WITH DATA is the default, it need not be specified.

```
DECLARE LOCAL TEMPORARY TABLE DepartmentsWithPhone
    AS ( SELECT D.*, E.Phone AS DepartmentPhone
        FROM Departments D JOIN Employees E
        ON D.DepartmentHeadID = E.EmployeeID )
    WITH DATA;
```

Related Information

Compound Statements
CREATE TABLE Statement [page 1002]

1.4.4.113 DELETE Statement

Deletes rows from the database.

Syntax

General use

```
DELETE [ row-limitation ]
[ FROM ] [ owner.]table-or-view [ [ AS ] correlation-name ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ], ... ]
[ OPTION( query-hint, ... ) ]
```

Transact-SQL

```
DELETE [ row-limitation ]
[ FROM ] [ owner.]table-or-view [ [ AS ] correlation-name ]
[ FROM table-expression ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ], ... ]
[ OPTION( query-hint, ... ) ]
```

```
table-or-view : identifier
```

```
row-limitation :
FIRST
| TOP { ALL | limit-expression } [ START AT startat-expression ]
limit-expression : simple-expression
startat-expression : simple-expression
simple-expression :
integer
| variable
| ( simple-expression )
| ( simple-expression { + | - | * } simple-expression )
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| FORCE NO OPTIMIZATION
| option-name = option-value
```

```
table-expression : a full table expression that can include joins
```

```
option-name : identifier
```

```
option-value :
hostvar (indicator allowed)
| string
| identifier
| number
```

Parameters

row-limitation clause

The row limitation clause allows you to restrict the rows being deleted to only a subset of the rows that satisfy the WHERE clause. The TOP and START AT arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The TOP argument must evaluate to a value greater than or equal to 0. The START AT argument must evaluate to a value greater than 0. When specifying these clauses, an ORDER BY clause is required to order the rows in a meaningful manner.

FROM clause

The FROM clause indicates the table from which rows will be deleted. In the Transact-SQL syntax, the second FROM clause in the DELETE statement determines the rows to be deleted from the specified table based on joins with other tables. `table-expression` can contain arbitrarily complex table expressions, including derived tables and KEY and NATURAL joins.

The following examples illustrate how correlation names are matched when the Transact-SQL syntax is used. With this statement:

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_2 AS alias_2
WHERE ...
```

table table_1 doesn't have a correlation name in the first FROM clause but does in the second FROM clause. In this case, table_1 in the first clause is identified with alias_1 in the second clause. There is only one instance of table_1 in this statement. This is allowed as an exception to the general rule that where a table is identified with a correlation name and without a correlation name in the same statement, two instances of the table are considered.

However, in the following example, there are two instances of table_1 in the second FROM clause. The statement fails with a syntax error because it is not clear which instance of the table_1 from the second FROM clause matches the first instance of table_1 in the first FROM clause.

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

WHERE clause

The DELETE statement deletes all the rows that satisfy the conditions in the WHERE clause. If no WHERE clause is specified, all rows from the named table are deleted. If a second FROM clause is present, the WHERE clause qualifies the rows of the second FROM clause's `table-expression`.

ORDER BY clause

Specifies the sort order for the rows to be deleted. Normally, the order in which rows are updated does not matter. However, with the FIRST or TOP clause, the order can be significant.

You cannot use ordinal column numbers in the ORDER BY clause.

Each item in the ORDER BY list can be labeled as ASC for ascending order (the default) or DESC for descending order.

OPTION clause

Use this clause to specify hints for executing the statement. The setting you specify is only applicable to the current statement and takes precedence over any public or temporary option settings, including those set by ODBC-enabled applications. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- `option-name` = `option-value`.
  Use an `OPTION( isolation_level = ... )` specification in the query text to override all other means of specifying isolation level for a query.
  Use an `OPTION( parameterization_level = ... )` specification in the query text to override the parameterization level for a query.

Remarks

Deleting a significant amount of data using the DELETE statement causes an update to column statistics.

To delete all of the rows of a table, consider using the more efficient TRUNCATE TABLE statement.

DELETE operations can be performed on views if the query specification defining the view is updatable. A view is updatable provided the SELECT statement defining the view has only one table in the FROM clause and does not contain a DISTINCT clause, a GROUP BY clause, a WINDOW clause, an aggregate function, or involve a set operator such as UNION or INTERSECT.

Privileges

You must be the table owner, or have SELECT and DELETE privileges on the table.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Core Feature. However, the following features are not in the standard:

- The optional FROM keyword.
- The `row-limitation` clause and the ORDER BY clause.
- The OPTION clause.

The Transact-SQL syntax is a Transact-SQL vendor extension.

Example

Remove all data before 2000 from the FinancialData table.

```
DELETE
FROM GROUPO.FinancialData
WHERE Year < 2000;
```

Remove the first 10 orders from SalesOrderItems table where ship date is older than 2001-01-01 and their region is Central.

```
DELETE TOP 10
```

```
FROM GROUPO.SalesOrderItems
FROM GROUPO.SalesOrders
WHERE SalesOrderItems.ID = SalesOrders.ID
  and ShipDate < '2001-01-01' and Region ='Central'
ORDER BY ShipDate ASC;
```

Remove department 600 from the database, executing the statement at isolation level 3.

```
DELETE FROM GROUPO.Departments
WHERE DepartmentID = 600
OPTION( isolation_level = 3 );
```

Related Information

Row Limitation Clauses in SELECT, UPDATE, and DELETE Query Blocks
Regular Views
Locks During Deletes
TRUNCATE Statement [page 1442]
INSERT Statement [page 1232]
INPUT Statement [Interactive SQL] [page 1224]
FROM Clause [page 1173]

1.4.4.114 DELETE Statement (Positioned) [ESQL] [SP]

Deletes the data at the current location of a cursor.

Syntax

```
DELETE [ [FROM ]table ] WHERE CURRENT OF cursor-name
```

```
cursor-name :  identifier | hostvar
```

```
table : [ owner.]table-or-view [ [ AS ] correlation-name ]
```

```
owner : identifier
```

```
table-or-view : identifier
```

```
correlation-name : identifier
```

Remarks

This form of the DELETE statement deletes the current row of the specified cursor. The current row is defined to be the last row fetched from the cursor.

The table from which rows are deleted is determined as follows:

- If no FROM clause is included, the cursor must be on a single table only.
- If the cursor is for a joined query (including using a view containing a join), then the FROM clause must be used. Only the current row of the specified table is deleted. The other tables involved in the join are not affected.
- If a FROM clause is included, `table` must unambiguously identify an updatable table in the cursor. If a`correlation-name` is specified, the server attempts to match that correlation name with a correlation name specified in the underlying cursor. If a correlation name is not specified in the DELETE statement, and a table owner is not specified, then the server attempts to match `table-or-view` with an updatable table in the underlying cursor. `table-or-view` is first matched against any correlation names.
  - If a correlation name exists in the underlying cursor, `table-or-view` may be matched with the corresponding correlation name.
  - If a correlation name does not exist, `table-or-view` must unambiguously match a table name in the cursor.
- If a FROM clause is included, and a table owner is specified, `table` must unambiguously match an updatable table in the underlying cursor.
- The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

Privileges

You must be the table owner, or have DELETE privilege on the table.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The DELETE statement (positioned) is a Core Feature. The ability to use a positioned DELETE statement from within an Embedded SQL program is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL".

The FROM keyword is mandatory in the ANSI/ISO SQL Standard, but optional in the software. The range of cursors that can be updated may contain extensions that are not in the standard if the ansi_update_constraints option is set to Off.

Example

The following statement removes the current row in the cursor cur_employee from the database.

```
DELETE
WHERE CURRENT OF cur_employee;
```

Related Information

UPDATE Statement [page 1463]
UPDATE (Positioned) Statement [ESQL] [SP] [page 1470]
INSERT Statement [page 1232]
PUT Statement [ESQL] [page 1313]

1.4.4.115 DESCRIBE Statement [ESQL]

Gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

Syntax

```
DESCRIBE
[ USER TYPES ]
[ ALL | BIND VARIABLES FOR | INPUT | OUTPUT
| SELECT LIST FOR ]
[ LONG NAMES [ long-name-spec ] | WITH VARIABLE RESULT ]
[ FOR ] { statement-name | CURSOR cursor-name }
INTO sqlda-name
```

```
long-name-spec :
OWNER.TABLE.COLUMN
| TABLE.COLUMN
| COLUMN
```

```
statement-name :  identifier | hostvar
```

```
cursor-name :  declared cursor
```

```
sqlda-name : identifier
```

Parameters

USER TYPES clause

A DESCRIBE statement with the USER TYPES clause returns information about domains of a column. Typically, such a DESCRIBE is done when a previous DESCRIBE returns an indicator of DT_HAS_USERTYPE_INFO.

The information returned is the same as for a DESCRIBE without the USER TYPES keywords, except that the sqlname field holds the name of the domain, instead of the name of the column.

If the DESCRIBE uses the LONG NAMES clause, the sqldata field holds this information.

ALL clause

DESCRIBE ALL allows you to describe INPUT and OUTPUT with one request to the database server. This has a performance benefit. The OUTPUT information is filled in the SQLDA first, followed by the INPUT information. The sqld field contains the total number of INPUT and OUTPUT variables. The DT_DESCRIBE_INPUT bit in the indicator variable is set for INPUT variables and clear for OUTPUT variables.

BIND VARIABLES FOR clause

Equivalent to the INPUT clause.

SELECT LIST FOR clause

Equivalent to the OUTPUT clause.

INPUT clause

A bind variable is a value supplied by the application when the database executes the statements. Bind variables can be considered parameters to the statement. DESCRIBE INPUT fills in the name fields in the SQLDA with the bind variable names. DESCRIBE INPUT also puts the number of bind variables in the sqlda field of the SQLDA.

DESCRIBE uses the indicator variables in the SQLDA to provide additional information. DT_PROCEDURE_IN and DT_PROCEDURE_OUT are bits that are set in the indicator variable when a CALL statement is described. DT_PROCEDURE_IN indicates an IN or INOUT parameter and DT_PROCEDURE_OUT indicates an INOUT or OUT parameter. Procedure RESULT columns will have both bits clear. After a describe OUTPUT, these bits can be used to distinguish between statements that have result sets (need to use OPEN, FETCH, RESUME, CLOSE) and statements that do not (need to use EXECUTE). DESCRIBE INPUT only sets DT_PROCEDURE_IN and DT_PROCEDURE_OUT appropriately when a bind variable is an argument to a CALL statement; bind variables within an expression that is an argument in a CALL statement will not set the bits.

OUTPUT clause

The DESCRIBE OUTPUT statement fills in the data type and length for each SELECT list item in the SQLDA. The name field is also filled in with a name for the SELECT list item. If an alias is specified for a SELECT list item, the name will be that alias. Otherwise, the name is derived from the SELECT list item: if the item is a simple column name, it is used; otherwise, a substring of the expression is used. DESCRIBE will also put the number of SELECT list items in the sqld field of the SQLDA.

If the statement being described is a UNION of two or more SELECT statements, the column names returned for DESCRIBE OUTPUT are the same column names which would be returned for the first SELECT statement.

If you describe a CALL statement, the DESCRIBE OUTPUT statement fills in the data type, length, and name in the SQLDA for each INOUT or OUT parameter in the procedure. DESCRIBE OUTPUT also puts the number of INOUT or OUT parameters in the sqld field of the SQLDA.

If you describe a CALL statement with a result set, the DESCRIBE OUTPUT statement fills in the data type, length, and name in the SQLDA for each RESULT column in the procedure definition. DESCRIBE OUTPUT will also put the number of result columns in the sqld field of the SQLDA.

LONG NAMES clause

The LONG NAMES clause is provided to retrieve column names for a statement or cursor. Without this clause, there is a 29-character limit on the length of column names; with the clause, names of an arbitrary length are supported.

If LONG NAMES is used, the long names are placed into the SQLDATA field of the SQLDA, as if you were fetching from a cursor. None of the other fields (SQLLEN, SQLTYPE, and so on) are filled in. The SQLDA must be set up like a FETCH SQLDA: it must contain one entry for each column, and the entry must be a string type. If there is an indicator variable, truncation is indicated in the usual fashion.

The default specification for the long names is TABLE.COLUMN.

WITH VARIABLE RESULT clause

This clause is used to describe procedures that can have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database server sets the SQLCOUNT value after the DESCRIBE statement to one of the following values:

0

The result set may change. The procedure call should be described again following each OPEN statement.

1

The result set is fixed. No re-describing is required.

Remarks

The DESCRIBE statement sets up the named SQLDA to describe either the OUTPUT (equivalently SELECT LIST FOR) or the INPUT (equivalently BIND VARIABLES FOR) for the named statement.

In the INPUT case, DESCRIBE BIND VARIABLES does not set up the data types in the SQLDA: this needs to be done by the application. The ALL keyword allows you to describe INPUT and OUTPUT in one SQLDA.

If you specify a statement name, the statement must have been previously prepared using the PREPARE statement with the same statement name and the SQLDA must have been previously allocated.

If you specify a cursor name, the cursor must have been previously declared and opened. The default action is to describe the OUTPUT. Only SELECT statements and CALL statements have OUTPUT. A DESCRIBE OUTPUT on any other statement, or on a cursor that is not a dynamic cursor, indicates no output by setting the sqld field of the SQLDA to zero.

In Embedded SQL, NCHAR, NVARCHAR and LONG NVARCHAR are described as DT_FIXCHAR, DT_VARCHAR, and DT_LONGVARCHAR, respectively, by default. If the db_change_nchar_charset function has been called, these data types are described as DT_NFIXCHAR, DT_NVARCHAR and DT_LONGNVARCHAR, respectively.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The DESCRIBE OUTPUT statement is optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The DESCRIBE INPUT statement is optional ANSI/ISO SQL Language Feature B032, "Extended dynamic SQL". Many of the other clauses of the DESCRIBE statement are not in the standard. These include:

- The USER TYPES, ALL, BIND VARIABLES FOR, LONG NAMES, and WITH VARIABLE RESULT clauses.
- DESCRIBE uses the INTO clause to identify the sqlda; in the ANSI/ISO SQL Standard, the USING keyword is used instead.
- In the ANSI/ISO SQL Standard, the CURSOR clause ends with the keyword STRUCTURE. STRUCTURE is not supported by the software.

Example

The following example shows how to use the DESCRIBE statement:

```
sqlda = alloc_sqlda( 3 );
EXEC SQL DESCRIBE OUTPUT
  FOR employee_statement
  INTO sqlda;
if( sqlda->sqld  >  sqlda->sqln ) {
  actual_size = sqlda->sqld;
  free_sqlda( sqlda );
  sqlda = alloc_sqlda( actual_size );
  EXEC SQL DESCRIBE OUTPUT
    FOR employee_statement
    INTO sqlda;
}
```

Related Information

The SQL Descriptor Area (SQLDA)

ALLOCATE DESCRIPTOR Statement [ESQL] [page 660]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
OPEN Statement [ESQL] [SP] [page 1286]
PREPARE Statement [ESQL] [page 1308]
db_change_nchar_charset Function
LONG NVARCHAR Data Type [page 134]
NCHAR Data Type [page 136]
NVARCHAR Data Type [page 138]

1.4.4.116 DESCRIBE Statement [Interactive SQL]

Returns information about a given database object.

Syntax

Describe database objects

```
DESCRIBE [ [ INDEX FOR ] TABLE | PROCEDURE ] [ owner.]object-name
```

```
 object-name :
table
| view
| materialized view
| procedure
| function
```

Describe the current connection

```
DESCRIBE CONNECTION
```

Parameters

INDEX FOR clause

Indicates that you want to see the indexes for the specified `object-name`.

TABLE clause

Indicates that `object-name` to be described is a table or a view.

PROCEDURE clause

Indicates that `object-name` is a procedure or a function.

Remarks

Use DESCRIBE TABLE to list all the columns in the specified table or view. The DESCRIBE TABLE statement returns one row per table column, containing:

Column

The name of the column.

Type

The type of data in the column.

Nullable

Whether nulls are allowed (1=yes, 0=no).

Primary Key

Whether the column is in the primary key (1=yes, 0=no).

Use DESCRIBE INDEX FOR TABLE to list all the indexes for the specified table. The DESCRIBE TABLE statement returns one row per index, containing:

Index Name

The name of the index.

Columns

The columns in the index.

Unique

Whether the index is unique (1=yes, 0=no).

Type

The type of index. Possible values are: Clustered, Statistic, Hashed, and Other.

Use DESCRIBE PROCEDURE to list all the parameters used by the specified procedure or function. The DESCRIBE PROCEDURE statement returns one row for each parameter, containing:

Parameter

The name of the parameter.

Type

The data type of the parameter.

In/Out

Information about what is passed to, or returned from, the parameter. Possible values are:

In

The parameter is passed to the procedure, but is not modified.

Out

The procedure ignores the parameter's initial value and sets its value when the procedure returns.

In/Out

The parameter is passed to the procedure and the procedure sets the parameter's value when the procedure returns.

Result

The parameter returns a result set.

Return

The parameter returns a declared return value.

When using DESCRIBE PROCEDURE, the returned parameter list is retrieved from the SYSPROCPARM system view.

If you do not specify either TABLE or PROCEDURE (for example, DESCRIBE `object-name`), Interactive SQL assumes the object is a table. However, if no such table exists, Interactive SQL attempts to describe the object as either a procedure or a function.

Use the DESCRIBE statement to list information about the database or database server that Interactive SQL is connected to. The following properties are returned:

Database Product

The name and version number of the database product Interactive SQL is connected to (for example, SQL Anywhere 17.0.11.1293).

Host Name

The network name of the computer the database server is running on.

Host TCP/IP Address

The IP address of the computer the database server is running on.

Host Operating System

The name and version number of the operating system used by the computer the database server is running on.

Server Name

The name of the database server.

Server TCP/IP Port

The port number used by the database server for the current connection.

Database Name

The name of the database that Interactive SQL is connected to.

Database Character Set

The character set used for CHAR columns in the database.

Connection String

The connection string that was used to connect to the database or database server. Three asterisks replace passwords.

Properties that do not apply to the current connection are omitted. For example, if you connect to a database server using shared memory, then the TCP/IP port is omitted.

Privileges

None

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Describe the columns in the Departments table:

```
DESCRIBE TABLE GROUPO.Departments;
```

Here is an example of the result set for this statement:

Column	Type	Nullable	Primary key
DepartmentID	integer	0	1
DepartmentName	char(40)	0	0
DepartmentHeadID	integer	1	0

List the indexes for the Customers table:

```
DESCRIBE INDEX FOR TABLE GROUPO.Customers;
```

Here is an example of the results for this statement:

Index Name	Columns	Unique	Type
IX_customer_name	Surname,GivenName	0	Clustered

Related Information

Interactive SQL
ALTER PROCEDURE Statement [page 704]

1.4.4.117 DETACH TRACING Statement (Deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The DETACH TRACING session ends a diagnostic tracing session.

Syntax

```
DETACH TRACING { WITH | WITHOUT } SAVE
```

Parameters

WITH SAVE clause

Specify WITH SAVE to save unsaved diagnostic data in the diagnostic tables.

WITHOUT SAVE clause

Specify WITHOUT SAVE if you do not want to save unsaved tracing data.

Remarks

Execute this statement from the database being profiled to stop sending diagnostic information to the diagnostic tables. If you specify the WITHOUT SAVE clause, then you can still save the data later, assuming that the tracing database is still running and another tracing session has not been started, by using the sa_save_trace_data system procedure.

To see the current tracing levels set for a database, look in the sa_diagnostic_tracing_level table.

Note

Tracing information is *not* unloaded as part of a database unload or reload operation. To transfer tracing information from one database to another, you must do so manually by copying the contents of the sa_diagnostic_* tables; however, this is not recommended.

Privileges

You must have the MANAGE PROFILING system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Diagnostic Tracing (Deprecated)
SQL Anywhere Profiler
ATTACH TRACING Statement (Deprecated) [page 775]
REFRESH TRACING LEVEL Statement (Deprecated) [page 1327]
sa_diagnostic_tracing_level Table (Deprecated) [page 1508]
sa_save_trace_data System Procedure [page 1693]

1.4.4.118 DISCONNECT Statement [ESQL] [Interactive SQL]

Drops a connection to a database.

Syntax

```
DISCONNECT [ connection-name | CURRENT | ALL ]
```

```
connection-name :
identifier
| string
| hostvar
```

Remarks

The DISCONNECT statement drops a connection with the database server and releases all resources used by it. If the connection to be dropped was named on the CONNECT statement, the name can be specified. Specifying ALL will drop all the application's connections to all database environments. CURRENT is the default, and drops the current connection.

Before closing the database connection, Interactive SQL automatically executes a COMMIT statement if the commit_on_exit option is set to On. If this option is set to Off, Interactive SQL performs an implicit ROLLBACK. By default, the commit_on_exit option is set to On.

This statement is not supported in procedures, triggers, events, or batches.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

DISCONNECT comprises optional ANSI/ISO SQL Language Feature F771. The ability to specify DISCONNECT without a parameter, and the commit_on_exit option, are not in the standard.

Example

The following statement shows how to use DISCONNECT in Embedded SQL:

```
EXEC SQL DISCONNECT :conn_name
```

The following statement shows how to use DISCONNECT from Interactive SQL to disconnect all connections:

```
DISCONNECT ALL;
```

Related Information

Interactive SQL
DROP CONNECTION Statement [page 1088]
CONNECT Statement [ESQL] [Interactive SQL] [page 815]
SET CONNECTION Statement [Interactive SQL] [ESQL] [page 1378]

1.4.4.119 DROP CERTIFICATE Statement

Drops a certificate from the database.

Syntax

```
DROP CERTIFICATE [ IF EXISTS ] certificate-name
```

Remarks

DROP CERTIFICATE deletes a certificate from the ISYSCERTIFICATE system table.

Use the IF EXISTS clause if you do not want an error returned when the DROP CERTIFICATE statement attempts to remove a certificate that does not exist.

Privileges

You must have the MANAGE CERTIFICATES system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

```
DROP CERTIFICATE mycert;
```

Related Information

CREATE CERTIFICATE Statement [page 820]

1.4.4.120 DROP CONNECTION Statement

Drops a user's connection to the database.

Syntax

```
DROP CONNECTION connection-id
```

Remarks

The DROP CONNECTION statement disconnects a user from the database by dropping the connection to the database.

The `connection-id` parameter is an integer constant. You can obtain the `connection-id` using the sa_conn_info system procedure.

This statement is not supported in procedures, triggers, events, or batches.

Privileges

You must have the DROP CONNECTION system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following procedure drops a connection identified by its connection number. When executing the DROP CONNECTION statement from within a procedure, you should do so using the EXECUTE IMMEDIATE statement, as shown in this example:

```
CREATE PROCEDURE drop_connection_by_id( IN conn_number INTEGER )
 BEGIN
   EXECUTE IMMEDIATE 'DROP CONNECTION ' || conn_number;
 END;
```

The following statement drops the connection with ID number 4.

```
DROP CONNECTION 4;
```

Related Information

Exception Handlers
CONNECT Statement [ESQL] [Interactive SQL] [page 815]
sa_conn_info System Procedure [page 1547]

1.4.4.121 DROP DATABASE Statement

Deletes all database files associated with a database.

Syntax

```
DROP DATABASE database-name [ KEY key ]
```

Remarks

The DROP DATABASE statement physically deletes all associated database files from disk. If the database file does not exist, or is not in a suitable condition for the database to be started, an error is generated.

DROP DATABASE cannot be used in stored procedures, triggers, events, or batches.

The database to be dropped must not be running when the DROP DATABASE statement is used. You cannot be connected to the database you are dropping. You must be connected to a different database. For example, connect to the utility database.

You must specify a key to drop a strongly encrypted database. The key can be either a string (constant literal) or a variable reference.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side Effects

In addition to deleting the database files from disk, any associated transaction log file or transaction log mirror file is deleted.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Execute the following statement to drop the database `temp.db`:

```
DROP DATABASE 'c:\temp\temp.db';
```

Execute the following statement to drop a database based on the constant literal, mykey:

```
DROP DATABASE 'temp.db' KEY 'mykey';
```

Execute the following statement to drop a database based on the variable reference, mykeyvar:

```
DROP DATABASE 'temp.db' KEY mykeyvar;
```

Related Information

Connecting to the Utility Database (Connect Window)
Erase Utility (dberase)
CREATE DATABASE Statement [page 821]
DatabaseKey (DBKEY) Connection Parameter
-gu Database Server Option

1.4.4.122 DROP DATATYPE Statement

Removes a data type from the database.

Syntax

```
DROP DATATYPE datatype-name
```

Remarks

It is recommended that you use DROP DOMAIN rather than DROP DATATYPE, as DROP DOMAIN is the syntax used in the ANSI/ISO SQL Standard. You cannot drop system-defined data types (such as MONEY or UNIQUEIDENTIFIERSTR) from a database.

Privileges

You must be the owner of the data type, or have the DROP DATATYPE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Domain support is optional ANSI/ISO SQL Language Feature F251. The DROP DATATYPE statement is not in the standard.

Example

The following example creates and then drops a datatype called PhoneNum:

```
CREATE DATATYPE PhoneNum CHAR(12) NULL;
DROP DATATYPE PhoneNum;
```

Related Information

DROP DOMAIN Statement [page 1093]
CREATE DOMAIN Statement [page 836]
ALTER DOMAIN Statement [page 673]

1.4.4.123 DROP DBSPACE Statement

Removes a dbspace from the database.

Syntax

```
DROP DBSPACE dbspace-name
```

Remarks

You must drop all tables in the dbspace before dropping the dbspace. You cannot use the DROP DBSPACE statement to drop the predefined dbspaces SYSTEM, TEMPORARY, TEMP, TRANSLOG, or TRANSLOGMIRROR.

DROP DBSPACE is prevented whenever the statement affects an object that is currently being used by another connection.

You must be the only connection to the database to execute this statement.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side Effects

Automatic commit, and causes an implicit checkpoint.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example drops a fictitious dbspace, MyDBSpace, from the database.

```
DROP DBSPACE MyDBSpace;
```

Related Information

Predefined Dbspaces
Dropping a Dbspace (SQL Central)
Dropping a Dbspace (SQL)
CREATE DBSPACE Statement [page 830]
ALTER DBSPACE Statement [page 670]

1.4.4.124 DROP DOMAIN Statement

Removes a domain (data type) from the database.

Syntax

```
DROP DOMAIN domain-name
```

Remarks

DROP DOMAIN is prevented if the data type is used in a table column, or in a procedure or function argument. You must change data types on all columns defined using the domain to drop the data type. It is recommended that you use DROP DOMAIN rather than DROP DATATYPE, as DROP DOMAIN is the syntax used in the ANSI/ISO SQL Standard. You cannot drop system-defined data types (such as MONEY or UNIQUEIDENTIFIERSTR) from a database.

Privileges

You must be the owner of the domain, or have the DROP DATATYPE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Domain support is optional SQL Language Feature F251.

Example

The following example creates and then drops the domain CustPhoneNumber.

```
CREATE DOMAIN CustPhoneNumber CHAR(12) NULL;
DROP DOMAIN CustPhoneNumber;
```

Related Information

CREATE DOMAIN Statement [page 836]
ALTER DOMAIN Statement [page 673]

1.4.4.125 DROP EVENT Statement

Drops an event from the database.

Syntax

```
DROP EVENT [ IF EXISTS ] event-name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP EVENT statement attempts to remove an event that does not exist.

Events are not owned so do not specify an owner (an owner specification is ignored).

Privileges

You must have either the MANAGE ANY EVENT or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example drops a fictitious example, MyEvent, from the database.

```
DROP EVENT MyEvent;
```

Related Information

CREATE EVENT Statement [page 847]
ALTER EVENT Statement [page 675]
TRIGGER EVENT Statement [page 1441]

1.4.4.126 DROP EXTERNLOGIN Statement

Drops an external login from the database.

Syntax

Drop an external login

```
DROP EXTERNLOGIN login-name TO remote-server
```

Drop an external login (include variables in syntax)

```
DROP EXTERNLOGIN USER string | variable SERVER string | variable
```

Parameters

DROP clause

Specify the local user login ID.

SERVER clause

Specify the name of the remote server. The local user's alternate login name and password for that server is the external login that is deleted.

Remarks

DROP EXTERNLOGIN deletes an external login from the database.

> **i** Note
>
> For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:
>
> - The variable does not exist
> - The contents of the variable are NULL
> - The variable exceeds the length allowed by the parameter
> - The data type of the variable does not match that required by the parameter

Privileges

You must have the MANAGE ANY USER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example drops the DBA external login to the fictitious remote server, sybase1.

```
DROP EXTERNLOGIN DBA TO sybase1;
```

Related Information

Dropping External Logins (SQL Central)
CREATE EXTERNLOGIN Statement [page 858]

1.4.4.127 DROP FUNCTION Statement

Removes a function from the database.

Syntax

```
DROP FUNCTION [ IF EXISTS ] [ owner.]function-name
```

Remarks

If you do not want an error returned when the DROP FUNCTION statement attempts to remove a function that does not exist, then use the IF EXISTS clause.

Privileges

You must be the owner of the function, or have the DROP ANY PROCEDURE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Core Feature. The IF EXISTS clause is not in the standard.

Example

This example drops the fictitious function, MyFunction, from the database.

```
DROP FUNCTION MyFunction;
```

Related Information

CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [External Call] [page 866]
CREATE FUNCTION Statement [Web Service] [page 874]
ALTER FUNCTION Statement [page 682]

1.4.4.128 DROP INDEX Statement

Removes an index from the database.

Syntax

```
DROP INDEX [ IF EXISTS ] { [ [ owner.]table-name.]index-name |
[ [ owner.]materialized-view-name. ]index-name }
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP INDEX statement attempts to remove an index that does not exist.

When you specify the IF EXISTS clause and the named table cannot be located, an error is returned.

DROP INDEX is prevented when the statement affects an object that is currently being used by another connection.

The DROP INDEX statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

To drop an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

To drop an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

Side Effects

Automatic commit. The DROP INDEX statement closes all cursors for the current connection.

If you use the DROP INDEX statement to drop an index on a local temporary table an error is returned indicating that the index could not be found. Use the DROP TABLE statement to drop a local temporary table. Indexes on local temporary tables are dropped automatically when the local temporary table goes out of scope.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example drops a fictitious index, MyIndex, from the database.

```
DROP INDEX MyIndex;
```

Related Information

Snapshot Isolation
CREATE INDEX Statement [page 886]
ALTER INDEX Statement [page 685]

1.4.4.129 DROP LDAP SERVER Statement

Drops an LDAP server configuration object.

Syntax

```
DROP LDAP SERVER ldapua-server-name
[ WITH DROP ALL REFERENCES ]
[ WITH SUSPEND ]
```

Parameters

WITH DROP ALL REFERENCES clause

Specify the DROP ALL REFERENCES clause to drop an LDAP server configuration object that is referenced from a login policy.

WITH SUSPEND clause

Specify the WITH SUSPEND clause to drop an LDAP server configuration object that is in a READY or ACTIVE state.

Remarks

This statement removes the LDAP server configuration object from the SYSLDAPSERVER system view after checking that the LDAP server configuration object is not in the READY or ACTIVE state. The statement fails when the state is READY or ACTIVE to ensure that the LDAP server is not in active use. To override this check, specify the WITH SUSPEND clause.

By default, a reference to the LDAP server configuration object in a login policy also causes this statement to fail. To remove an LDAP server configuration object that is referenced in a login policy, add the DROP ALL REFERENCES clause. Adding DROP ALL REFERENCES does not remove the reference from the login policy; it allows you to drop the configuration object when there are references. You must still remove the reference to the LDAP server configuration object from the login policy.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example suspends an LDAP server configuration object named apps_primary that is referenced in a login policy, and then drops it.

```
DROP LDAP SERVER apps_primary WITH DROP ALL REFERENCES WITH SUSPEND;
```

Related Information

CREATE LDAP SERVER Statement [page 891]
ALTER LDAP SERVER Statement [page 687]
VALIDATE LDAP SERVER Statement [page 1479]
ALTER LOGIN POLICY Statement [page 690]

1.4.4.130 DROP LOGIN POLICY Statement

Drops a login policy.

Syntax

DROP LOGIN POLICY `policy-name`

Parameters

policy-name

The name of the login policy.

Remarks

The statement fails if you drop a policy that is assigned to a user. You cannot drop the root login policy. Use the ALTER USER statement to change a user's policy assignment.

Privileges

You must have the MANAGE ANY LOGIN POLICY system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates a login policy, Test11, and then deletes it.

```
CREATE LOGIN POLICY Test11;
DROP LOGIN POLICY Test11;
```

Related Information

Login Policies
Deleting a Login Policy
ALTER LOGIN POLICY Statement [page 690]
ALTER USER Statement [page 769]
COMMENT Statement [page 808]
CREATE LOGIN POLICY Statement [page 894]
CREATE USER Statement [page 1045]
DROP USER Statement [page 1143]

1.4.4.131 DROP MATERIALIZED VIEW Statement

Removes a materialized view from the database.

Syntax

```
DROP MATERIALIZED VIEW [ IF EXISTS ] [ owner.]materialized-view-name
```

Remarks

All data in the table is automatically deleted as part of the dropping process. All indexes and keys for the materialized view are dropped as well.

Use the IF EXISTS clause if you do not want an error returned when the DROP MATERIALIZED VIEW statement attempts to remove a materialized view that does not exist.

You cannot execute a DROP MATERIALIZED VIEW statement on an object that is currently being used by another connection.

Executing a DROP MATERIALIZED VIEW statement changes the status of all dependent regular views to INVALID. To determine view dependencies before dropping a materialized view, use the sa_dependent_views system procedure.

Privileges

You must be the owner of the materialized view, or have the DROP ANY MATERIALIZED VIEW or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit occurs for DROP MATERIALIZED VIEW. Commit occurs for DROP MATERIALIZED VIEW IF EXISTS only when the view exists.

If the materialized view had been populated, DROP MATERIALIZED VIEW will trigger an automatic checkpoint. Closes all cursors for the current connection.

When a view is dropped, all procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the fact that the view does not exist. The unloading and loading of procedures and triggers can affect performance if you are regularly dropping and creating views.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example drops a fictitious materialized view, MyMaterializedView, from the database.

```
DROP MATERIALIZED VIEW MyMaterializedView;
```

Related Information

Advanced: Status and Properties for Materialized Views
CREATE MATERIALIZED VIEW Statement [page 896]
ALTER MATERIALIZED VIEW Statement [page 692]
REFRESH MATERIALIZED VIEW Statement [page 1321]
Materialized Views Restrictions
sa_dependent_views System Procedure [page 1571]

1.4.4.132 DROP MESSAGE Statement

Removes a message from the database.

Syntax

```
DROP MESSAGE msgnum
```

Remarks

None.

Privileges

You must be owner, or have the DROP MESSAGE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

DROP MESSAGE supplies the functionality provided by the sp_dropmessage() system procedure in Adaptive Server Enterprise.

Example

The following example creates and then drops a new message. To run this example, you must also have the CREATE MESSAGE system privilege:

```
CREATE MESSAGE 20000 AS 'End of line reached';
DROP MESSAGE 20000;
```

Related Information

PRINT Statement [T-SQL] [page 1312]
CREATE MESSAGE Statement [T-SQL] [page 899]
SYSUSERMESSAGE System View [page 1968]

1.4.4.133 DROP MIRROR SERVER Statement

Drops a mirror server.

Syntax

```
DROP MIRROR SERVER mirror-server-name
```

Remarks

Removes the specified mirror server definition from the database.

The mirror database stops. If the mirror database is the only database running on the server, then the server also stops.

Read-only scale-out and database mirroring each require a separate license.

Privileges

You must have the MANAGE ANY MIRROR SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example creates, and then drops, a mirror server named scaleout_primary2:

```
CREATE MIRROR SERVER "scaleout_primary2"
   AS PRIMARY
   connection_string = 'server=scaleout_primary1;host=winxp-2:6871,winxp-3:6872';
DROP MIRROR SERVER "scaleout_primary2";
```

Related Information

Database Mirroring
Separately Licensed Components
CREATE MIRROR SERVER Statement [page 901]
ALTER MIRROR SERVER Statement [page 695]
COMMENT Statement [page 808]

1.4.4.134 DROP MUTEX Statement

Drops the specified mutex.

Syntax

```
DROP MUTEX [ IF EXISTS ] [ owner.]mutex-name
```

Parameters

owner

The owner of the mutex. `owner` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

mutex-name

The name of the mutex. `mutex-name` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

IF EXISTS clause

Use this clause to drop a mutex only if it exists. If a mutex does not exist and this clause is specified, then nothing happens and no error is returned.

Remarks

If the mutex is locked by another connection, the drop operation proceeds without blocking but the mutex will persist in the namespace until the mutex is released. Connections waiting on the mutex receive an error immediately indicating that the object has been dropped.

Privileges

You must own the permanent mutex or have the DROP ANY MUTEX SEMAPHORE system privilege.

Side Effects

Automatic commit, but only for permanent mutexes.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement drops the protect_my_cr_section mutex:

```
DROP MUTEX protect_my_cr_section;
```

Related Information

Mutexes and Semaphores
CREATE MUTEX Statement [page 906]
LOCK MUTEX Statement [page 1265]
RELEASE MUTEX Statement [page 1329]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.135 DROP ODATA PRODUCER Statement

Drops an OData Producer.

Syntax

DROP ODATA PRODUCER [IF EXISTS] `name`

Remarks

Removes the definition of the OData Producer and all of its options from the system tables. Use this statement only when you want to delete the configuration data of the OData Producer. If you want to temporarily disable the OData Producer, then execute the following statement:

```
ALTER ODATA PRODUCER name NOT ENABLED;
```

Use the IF EXISTS clause if you do not want an error returned when the DROP ODATA PRODUCER statement attempts to remove an OData Producer that does not exist.

Privileges

You must have the MANAGE ODATA system privilege.

Side Effects

Automatic commit.

If the OData server is running, then the specified producer is shut down.

Example

To drop the OData Producer named prod, execute the following statement:

```
DROP ODATA PRODUCER prod;
```

Related Information

OData Support
Network Protocol Options
ALTER ODATA PRODUCER Statement [page 699]
CREATE ODATA PRODUCER Statement [page 908]
COMMENT Statement [page 808]
-xs Database Server Option
SYSODATAPRODUCER System View [page 1929]

1.4.4.136 DROP PROCEDURE Statement

Removes a procedure from the database.

Syntax

```
DROP PROCEDURE [ IF EXISTS ] [ owner.]procedure-name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP PROCEDURE statement attempts to remove a procedure that does not exist.

Privileges

You must be the owner of the procedure, or have the DROP ANY PROCEDURE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Core Feature. The IF EXISTS clause is not in the standard.

Example

This example creates a procedure called NewDepartment, and then drops it. To run this example, you must also have the CREATE PROCEDURE privilege.

```
CREATE PROCEDURE NewDepartment(
   IN id INT,
   IN name CHAR(35),
   IN head_id INT )
BEGIN
   INSERT
   INTO GROUPO.Departments ( DepartmentID,
       DepartmentName, DepartmentHeadID )
   VALUES ( id, name, head_id );
END;
DROP PROCEDURE NewDepartment;
```

Related Information

CREATE PROCEDURE Statement [page 913]

CREATE PROCEDURE Statement [External Call] [page 921]
CREATE PROCEDURE Statement [Web Service] [page 931]
ALTER PROCEDURE Statement [page 704]

1.4.4.137 DROP PUBLICATION Statement [MobiLink] [SQL Remote]

Drops a publication.

Syntax

```
DROP PUBLICATION [ IF EXISTS ] [ owner.]publication-name
```

```
owner, publication-name : identifier
```

Remarks

This statement is applicable only to MobiLink and SQL Remote.

In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

Use the IF EXISTS clause if you do not want an error returned when the DROP PUBLICATION statement attempts to remove a publication that does not exist.

DROP PUBLICATION requires exclusive access to all tables referred to in the publication.

Privileges

You must be the owner of the publication, or have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit. All subscriptions to the publication are dropped.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement drops the pub_contact publication.

```
DROP PUBLICATION pub_contact;
```

Related Information

Publications
Dropping a Publication
Dropping a Publication (SQL Central)
ALTER PUBLICATION Statement [MobiLink] [SQL Remote] [page 706]
CREATE PUBLICATION Statement [MobiLink] [SQL Remote] [page 946]

1.4.4.138 DROP REMOTE CONNECTION Statement

Drops remote data access connections to a remote server.

Syntax

```
DROP REMOTE CONNECTION TO server-name
CLOSE { CURRENT | ALL | connection-id }
[ FOR { EFFECTIVE USER | LOGIN USER | USER user-name } ]
```

Parameters

server-name

The remote data access server that was specified in the CREATE SERVER statement.

CLOSE clause

CLOSE CURRENT drops remote connections for the current local connection.

CLOSE ALL drops remote connections for all local connections.

CLOSE `connection-id` drops remote connections for the local connection with the specified ID.

FOR clause

FOR EFFECTIVE USER drops remote connections that were created with the current effective user's externlogin credentials.

FOR LOGIN USER drops remote connections that were created with the current login user's externlogin credentials.

FOR USER `user-name` drops remote connections that were created with the externlogin credentials for `user-name`.

If the FOR clause is omitted, then remote connections for all users are dropped.

Remarks

The DROP REMOTE CONNECTION statement allows you to explicitly close connections to a remote server. You may find this useful when a remote connection becomes inactive or is no longer needed.

Privileges

You must have the SERVER OPERATOR system privilege.

Side Effects

None

Example

Drop all remote connections, whether they are the current connection or not, to the myServer server for the effective user:

```
DROP REMOTE CONNECTION TO myServer CLOSE ALL FOR EFFECTIVE USER;
```

Drop the remote connection to the myServer server for the current local connection for user2:

```
DROP REMOTE CONNECTION TO myServer CLOSE CURRENT FOR USER user2;
```

Drop all remote connections, whether they are the current connection or not, to the myServer server for user2:

```
DROP REMOTE CONNECTION TO myServer CLOSE ALL FOR USER user2;
```

Drop the remote connection to the myServer server for the current local connection with the current connection's login user:

```
DROP REMOTE CONNECTION TO myServer CLOSE CURRENT FOR LOGIN USER;
```

Drop the remote connection to the myServer server for the connection with the ID connection-id and the current effective user:

```
DROP REMOTE CONNECTION TO myServer CLOSE connectionId FOR EFFECTIVE USER;
```

Related Information

CREATE SERVER Statement [page 962]
ALTER SERVER Statement [page 714]

1.4.4.139 DROP REMOTE [MESSAGE] Statement [SQL Remote]

Deletes a message type definition from a database.

Syntax

```
DROP REMOTE [ MESSAGE ]
TYPE message-system
```

```
message-system : FILE | FTP | HTTP | SMTP
```

Remarks

The statement removes a message type from a database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement drops the FILE message type from a database.

```
DROP REMOTE MESSAGE TYPE FILE;
```

The following statement drops the SMTP message type from a database. The word MESSAGE is optional.

```
DROP REMOTE TYPE SMTP;
```

Related Information

SQL Remote Message Systems
Deleting a Message Type (SQL Central)
CREATE REMOTE [MESSAGE] Statement [SQL Remote] [page 950]

1.4.4.140 DROP ROLE Statement

Removes a role from the database, or converts a user-extended role back to a regular user.

Syntax

```
DROP ROLE [ FROM USER ] role-name
[ WITH { REVOKE | DROP OBJECTS } ]
```

Parameters

role-name

Specify the name of the role you are dropping or converting.

FROM USER clause

Specify this clause to convert a user-extended role back to a regular user. The user retains any login privileges, system privileges, and roles they had.

WITH REVOKE clause

Specify WITH REVOKE when there are other users who have been granted `role-name`.

WITH DROP OBJECTS clause

Specify WITH DROP OBJECTS to drop the objects owned by `role-name`. If any of the objects cannot be dropped, for example because the object is currently in use, then the statement returns an error. You cannot specify this clause if `role-name` is a user-extended role.

Remarks

A user-defined role can be dropped from the database, and a user-extended role can be converted back to a regular user, as long as all dependent roles meet the minimum required number of administrative users with active passwords, as set by the min_role_admin database option.

When you convert a user-extended role back to a regular user, ownership of objects remains with the user that is being converted back to a regular user.

When you convert a user-extended role back to a regular user, any privileges that were granted to `role-name` remain with the user after they have been converted.

If you convert a user-extended role back to a regular user and any other roles and/or users were granted the user-extended role, the WITH REVOKE clause must be specified or else the statement returns an error message and fails.

If any objects impacted by the drop operation are in use, the statement returns an error message and the statement fails.

Privileges

You must have administrative rights for the role being dropped.

Side Effects

Automatic commit

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement converts a user-extended role named Joe back to a regular user. Objects owned by the user-extended role are now owned by the regular user, Joe. Users or roles that had been granted Joe retain the underlying privileges associated with the role.

```
DROP ROLE FROM USER Joe;
```

The following statement drops a user-extended role named Jack from the database. If the role Jack owned any objects, ownership of the object reverts to user Jack. Users or roles that were granted Jack retains the underlying privileges associated with the role Jack.

```
DROP ROLE Jack;
```

The following statement converts a user-extended role named Sam back to a regular user. Users and roles who had been granted Sam has the privileges of Sam revoked.

```
DROP ROLE FROM USER Sam WITH REVOKE;
```

The following statement drops a role named Sales1. Users or roles that were granted Sales1 retain the underlying privileges associated with the Sales1.

```
DROP ROLE Sales1;
```

The following statement drops a role named Sales2. Users or roles that had been granted Sales2 lose all underlying privileges associated with Sales2.

```
DROP ROLE Sales2 WITH REVOKE;
```

The following statement converts a user-extended role named Marketing1 to a regular user named Marketing1, and drops any objects that it owned.

```
DROP ROLE FROM USER Marketing1 WITH DROP OBJECTS;
```

The following statement drops a role named Marketing2, drops the objects it owned, and revokes its underlying system privileges from those who had been granted the role.

```
DROP ROLE Marketing2 WITH REVOKE WITH DROP OBJECTS;
```

Related Information

User-extended Roles
ALTER ROLE Statement [page 710]
CREATE ROLE Statement [page 952]
min_role_admins Option

1.4.4.141 DROP SEMAPHORE Statement

Drops a semaphore.

Syntax

```
DROP SEMAPHORE [ IF EXISTS ] [ owner.]semaphore-name
```

Parameters

owner

The owner of the semaphore. `owner` can also be specified using an indirect identifier (for example, `'[@variable-name]'`).

semaphore-name

The name of the semaphore. `semaphore-name` can also be specified using an indirect identifier (for example, `'[@variable-name]'`).

IF EXISTS clause

Use this clause to drop a semaphore only if it exists. If a semaphore does not exist and this clause is specified, then nothing happens and no error is returned.

Remarks

An error is returned to any connection that is waiting to decrement the semaphore.

Privileges

You must have the DROP ANY MUTEX SEMAPHORE or DROP ANY OBJECT system privilege, or be the owner of the semaphore. For a temporary semaphore, you must be the connection that created the mutex.

Side Effects

Automatic commit, but only for permanent semaphores.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement drops a semaphore called license_counter:

```
DROP SEMAPHORE license_counter;
```

Related Information

Mutexes and Semaphores
CREATE SEMAPHORE Statement [page 957]
NOTIFY SEMAPHORE Statement [page 1282]
WAITFOR SEMAPHORE Statement [page 1483]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.142 DROP SEQUENCE Statement

Drops a sequence.

Syntax

```
DROP SEQUENCE [ owner.] sequence-name
```

Remarks

If the named sequence cannot be located, an error message is returned. When you drop a sequence, all synonyms for the name of the sequence are dropped automatically by the database server.

Privileges

You must be the owner of the sequence, or have the DROP ANY SEQUENCE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Sequences comprise optional ANSI/ISO SQL Language Feature T176.

Example

The following example creates and then drops a sequence named Test:

```
CREATE SEQUENCE Test
START WITH 4
INCREMENT BY 2
NO MAXVALUE
NO CYCLE
CACHE 15;
DROP SEQUENCE Test;
```

Related Information

Use of a Sequence to Generate Unique Values
ALTER SEQUENCE Statement [page 712]
CREATE SEQUENCE Statement [page 959]

1.4.4.143 DROP SERVER Statement

Drops a remote server from the catalog.

Syntax

Drop a remote server

```
DROP [ REMOTE ] SERVER server-name
```

Drop a remote server (SAP HANA syntax)

```
DROP REMOTE SOURCE remote-source-name
```

Parameters

The DROP SERVER, DROP REMOTE SERVER, and DROP REMOTE SOURCE statements are semantically equivalent. The DROP REMOTE SOURCE syntax is provided for compatibility with SAP HANA.

Remarks

You must drop all the proxy tables that have been defined for the remote server before this statement succeeds.

Privileges

You must have the SERVER OPERATOR system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example drops a fictitious server named ase_prod:

```
DROP SERVER ase_prod;
```

Related Information

Remote Data Access
Remote Servers and Remote Table Mappings
Directory Access Servers

ALTER SERVER Statement [page 714]
CREATE SERVER Statement [page 962]
DROP REMOTE CONNECTION Statement [page 1112]

1.4.4.144 DROP SERVICE Statement

Drops a web service.

Syntax

```
DROP SERVICE service-name
```

Remarks

This statement deletes a web service listed in the ISYSWEBSERVICE system table.

Privileges

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following SQL statement drops a fictitious web service named WebServiceTable:

```
DROP SERVICE WebServiceTable;
```

Related Information

CREATE SERVICE Statement [HTTP Web Service] [page 969]
CREATE SERVICE Statement [SOAP Web Service] [page 975]
ALTER SERVICE Statement [HTTP Web Service] [page 718]
ALTER SERVICE Statement [SOAP Web Service] [page 724]
SYSWEBSERVICE System View [page 1972]

1.4.4.145 DROP SPATIAL REFERENCE SYSTEM Statement

Drops a spatial reference system.

Syntax

```
DROP SPATIAL REFERENCE SYSTEM [ IF EXISTS ] name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP SPATIAL REFERENCE SYSTEM statement attempts to remove a spatial reference system that does not exist.

Privileges

You must be the owner, or have the MANAGE ANY SPATIAL OBJECT or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example drops a fictitious spatial reference system named Test.

```
DROP SPATIAL REFERENCE SYSTEM Test;
```

Related Information

Spatial Data
CREATE SPATIAL REFERENCE SYSTEM Statement [page 981]
ALTER SPATIAL REFERENCE SYSTEM Statement [page 730]

1.4.4.146 DROP SPATIAL UNIT OF MEASURE Statement

Drops a spatial unit of measurement.

Syntax

```
DROP SPATIAL UNIT OF MEASURE [ IF EXISTS ] identifier
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP SPATIAL UNIT OF MEASURE statement attempts to remove a spatial unit of measure that does not exist.

Privileges

You must be the owner of the spatial unit of measure, or have the MANAGE ANY SPATIAL OBJECT or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example drops a fictitious spatial unit of measure named Test.

```
DROP SPATIAL UNIT OF MEASURE Test;
```

Related Information

Spatial Data
CREATE SPATIAL UNIT OF MEASURE Statement [page 989]

1.4.4.147 DROP STATEMENT Statement [ESQL]

Frees statement resources.

Syntax

```
DROP STATEMENT [ owner.]statement-name
```

```
statement-name :
identifier
| hostvar
```

Remarks

The DROP STATEMENT statement frees resources used by the named prepared statement. These resources are allocated by a successful PREPARE statement, and are normally not freed until the database connection is released.

To drop the statement, you must first have prepared the statement.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard. In the ANSI/ISO SQL Standard, this functionality is provided by the DEALLOCATE PREPARE statement, which is part of the optional ANSI/ISO SQL Language Feature B032, "Extended dynamic SQL".

Example

The following are examples of DROP STATEMENT use:

```
EXEC SQL DROP STATEMENT S1;
EXEC SQL DROP STATEMENT :stmt;
```

Related Information

PREPARE Statement [ESQL] [page 1308]

1.4.4.148 DROP STATISTICS Statement

Erases all column statistics on the specified columns.

Syntax

```
DROP STATISTICS [ ON ] [owner.]object-name [ ( column-list ) ]
```

```
object-name :
table-name
| materialized-view-name
```

```
| temp-table-name
```

Remarks

The database server optimizer uses column statistics to determine the best strategy for executing each statement. The database server automatically gathers and updates these statistics. Column statistics are stored permanently in the database in the ISYSCOLSTAT system table. Column statistics gathered while processing one statement are available when searching for efficient ways to execute subsequent statements.

Occasionally, the column statistics can become inaccurate or relevant statistics may be unavailable. This condition is most likely to arise when few queries have been executed since a large amount of data was added, updated, or deleted.

The DROP STATISTICS statement deletes all internal statistical data from the ISYSCOLSTAT system table for the specified columns. This drastic step leaves the optimizer with no access to essential statistical information. Without these statistics, the optimizer can generate inefficient data access plans, causing poor database performance.

The DROP STATISTICS statement requires an exclusive lock on the table against which it is being performed. Execution of the statement cannot proceed until all other connections that refer to the table have either committed or rolled back the referring transactions, or closed any open cursors that refer to the table.

This statement should be used only during problem determination or when reloading data into a database that differs substantially from the original data.

The CREATE STATISTICS and DROP STATISTICS statements do not require or benefit from a commit with regards to the statistics, and only DROP STATISTICS has a commit as a side effect. However, the commit does not save changes to statistics (the statistics governor looks after that). Thus, while DROP STATISTICS does have a side effect of an automatic commit, it commits everything except for the dropping of the statistics.

Privileges

You must be the table owner, or have the MANAGE ANY STATISTICS or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Optimizer Estimates and Statistics
CREATE STATISTICS Statement [page 991]
SYSCOLSTAT System View [page 1902]

1.4.4.149 DROP SUBSCRIPTION Statement [SQL Remote]

Drops a subscription for a user from a publication.

Syntax

```
DROP SUBSCRIPTION TO publication-name [ ( subscription-value ) ]
 FOR subscriber-id, ...
```

```
subscription-value : string
```

```
subscriber-id : string
```

Parameters

publication-name

The name of the publication to which the user is being subscribed. This can include the owner of the publication.

subscription-value

A string that is compared to the subscription expression of the publication. This value is required if and only if the subscription you are dropping contains a subscription value.

subscriber-id

The user ID of the subscriber to the publication.

Remarks

Drops a SQL Remote subscription for a user ID to a publication in the current database. The user ID will no longer receive updates when data in the publication is changed.

In SQL Remote, publications and subscriptions are two-way relationships. If you drop a subscription for a remote user to a publication on a consolidated database, you should also drop the subscription for the consolidated database on the remote database to prevent updates on the remote database being sent to the consolidated database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement drops a subscription for the SamS user ID to the pub_contact publication.

```
DROP SUBSCRIPTION TO pub_contact
FOR Sam_Singer;
```

Related Information

CREATE SUBSCRIPTION Statement [SQL Remote] [page 993]
STOP SUBSCRIPTION Statement [SQL Remote] [page 1428]
SYSSUBSCRIPTION System View [page 1947]

1.4.4.150 DROP SYNCHRONIZATION PROFILE Statement [MobiLink]

Deletes a SQL Anywhere synchronization profile.

Syntax

```
DROP SYNCHRONIZATION PROFILE [ IF EXISTS ] name
```

Parameters

name

The name of the synchronization profile to delete.

Remarks

Synchronization profiles are named collections of synchronization options that can be used to control synchronization. Use the IF EXISTS clause if you do not want an error returned when the DROP SYNCHRONIZATION PROFILE statement attempts to remove a synchronization profile that does not exist.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

CREATE SYNCHRONIZATION PROFILE Statement [MobiLink] [page 996]
ALTER SYNCHRONIZATION PROFILE Statement [MobiLink] [page 736]

1.4.4.151 DROP SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

Drops a synchronization subscription in a remote database.

Syntax

```
DROP SYNCHRONIZATION SUBSCRIPTION { subscription-name |
TO publication-name
[ FOR ml-username, ... ] }
```

Parameters

subscription-name

Specifies the name of the subscription to drop.

TO clause

Specifies the name of a publication.

FOR clause

Specifies one more users.

Omitting this clause drops the default settings for the publication.

Remarks

Requires exclusive access to all tables referred to in the publication.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example drops the subscription named sales:

```
DROP SYNCHRONIZATION SUBSCRIPTION sales;
```

The following example drops the subscription between the MobiLink user SSinger and the publication called sales_publication:

```
DROP SYNCHRONIZATION SUBSCRIPTION
   TO user.sales_publication
   FOR "SSinger";
```

The following example omits the FOR clause, and so drops the default settings for the publication called sales_publication:

```
DROP SYNCHRONIZATION SUBSCRIPTION
   TO user.sales_publication;
```

Related Information

Dropping MobiLink Subscriptions
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 737]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink] [page 997]
SYSSYNC System View [page 1948]

1.4.4.152 DROP SYNCHRONIZATION USER Statement [MobiLink]

Drops one or more synchronization users from a SQL Anywhere remote database.

Syntax

```
DROP SYNCHRONIZATION USER ml-username, ...
```

```
ml-username : identifier
```

Remarks

Drop one or more synchronization users from a MobiLink remote database.

You must have exclusive access to all tables referred to by publications subscribed to by the user.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

All subscriptions associated with the user are also deleted.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Remove MobiLink user SSinger from the database.

```
DROP SYNCHRONIZATION USER SSinger;
```

Related Information

ALTER SYNCHRONIZATION USER Statement [MobiLink] [page 740]
CREATE SYNCHRONIZATION USER Statement [MobiLink] [page 1000]
SYSSYNC System View [page 1948]

1.4.4.153 DROP TABLE Statement

Removes a table from the database.

Syntax

```
DROP TABLE [ IF EXISTS ] [ owner.]table-name
```

Remarks

When you remove a table, all data in the table is automatically deleted as part of the dropping process. All indexes and keys for the table are dropped as well.

Use the IF EXISTS clause if you do not want an error returned when the DROP TABLE statement attempts to remove a table that does not exist.

You cannot execute a DROP TABLE statement when the statement affects a table that is currently being used by another connection. Execution of a DROP TABLE statement is also prevented if there is a materialized view dependent on the table.

When you execute a DROP TABLE statement, the status of all dependent regular views change to INVALID. To determine view dependencies before dropping a table, use the sa_dependent_views system procedure.

Global temporary tables cannot be dropped unless all users that have referenced the temporary table have disconnected.

Privileges

You must be the owner of the table, or have the DROP ANY TABLE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit occurs for DROP TABLE. Commit occurs for DROP TABLE IF EXISTS only when the table exists.

DROP TABLE may also cause an automatic checkpoint. Executing a DROP TABLE statement closes all cursors for the current connection.

You can use the DROP TABLE statement to drop a local temporary table.

Standards

ANSI/ISO SQL Standard

Core Feature, however, the IF EXISTS clause is not in the standard. The ability to drop a declared local temporary table with the DROP TABLE statement is not in the standard.

Example

This example drops the MyTable table from the database.

```
DROP TABLE MyTable;
```

This example drops the MyTable table from the database. Because IF EXISTS is specified, if the table does not exist, an error is *not* returned and no commit occurs.

```
DROP TABLE IF EXISTS MyTable;
```

Related Information

Dropping a Table
CREATE TABLE Statement [page 1002]
ALTER TABLE Statement [page 742]
sa_dependent_views System Procedure [page 1571]

1.4.4.154 DROP TEXT CONFIGURATION Statement

Drops a text configuration object.

Syntax

DROP TEXT CONFIGURATION `[ owner.]text-config-name`

Remarks

Attempting to drop a text configuration object with dependent text indexes results in an error. You must drop the dependent text indexes before dropping the text configuration object.

Text configuration objects are stored in the ISYSTEXTCONFIG system table.

Privileges

You must be the owner of the text configuration object, or have the DROP ANY TEXT CONFIGURATION or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements create and drop the mytextconfig text configuration object:

```
CREATE TEXT CONFIGURATION mytextconfig FROM default_char;
DROP TEXT CONFIGURATION mytextconfig;
```

Related Information

Full Text Search
Text Configuration Object Concepts and Reference
DROP TEXT INDEX Statement [page 1136]
SYSTEXTCONFIG System View [page 1958]
CREATE TEXT CONFIGURATION Statement [page 1028]
ALTER TEXT CONFIGURATION Statement [page 757]

1.4.4.155 DROP TEXT INDEX Statement

Removes a text index from the database.

Syntax

DROP TEXT INDEX `text-index-name`

```
ON [ owner.]table-name
```

Parameters

ON clause

Use this clause to specify the table or materialized view on which the text index was built.

Remarks

You must drop dependent text indexes before you can drop a text configuration object.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

To drop a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

To drop a text index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements create and drop the TextIdx text index:

```
CREATE TEXT INDEX TextIdx ON GROUPO.MarketingInformation ( Description )
DROP TEXT INDEX TextIdx ON GROUPO.MarketingInformation;
```

Related Information

Full Text Search
Text Index Concepts and Reference
Snapshot Isolation
SYSTEXTCONFIG System View [page 1958]
CREATE TEXT INDEX Statement [page 1030]
ALTER TEXT INDEX Statement [page 762]
REFRESH TEXT INDEX Statement [page 1325]
TRUNCATE TEXT INDEX Statement [page 1445]

1.4.4.156 DROP TIME ZONE Statement

Drops a time zone from the database.

Syntax

```
DROP TIME ZONE [ IF EXISTS ] name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP TIME ZONE statement attempts to remove a time zone that does not exist.

A time zone cannot be dropped if it has been set using the time_zone database option.

Privileges

You must have either the MANAGE TIME ZONE or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit.

Example

This example drops a fictitious time zone, MyTimeZone, from the database.

```
DROP TIME ZONE MyTimeZone;
```

Related Information

ALTER TIME ZONE Statement [page 764]
COMMENT Statement [page 808]
CREATE TIME ZONE Statement [page 1033]
time_zone Option
SYSTIMEZONE System View [page 1961]

1.4.4.157 DROP TRACE EVENT Statement

Drops a user-defined trace event.

Syntax

```
DROP TRACE EVENT [ IF EXISTS ] trace-event-name
```

Remarks

This statement only drops user-defined trace events. If you do not want an error returned when the DROP TRACE EVENT statement attempts to remove a trace event that does not exist, use the IF EXISTS clause. If one ore more event tracing sessions reference the trace event, then the trace event cannot be dropped until all the referencing trace sessions are dropped.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Drop the trace event named my_event:

```
DROP TRACE EVENT my_event;
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_events System Procedure [page 1851]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.4.4.158 DROP TRACE EVENT SESSION Statement

Drops a trace event session.

Syntax

```
DROP TRACE EVENT SESSION [ IF EXISTS ] session-name UNCONDITIONALLY
[ ON SERVER ]
```

Remarks

When UNCONDITIONALLY is specified, dropping an active session automatically stops the session before removing its definition. Otherwise, an error is returned.

Specify the ON SERVER clause to drop a trace event session that is recording trace events from all databases on the database server. If this clause is not specified, then the trace event session on the current database is deleted.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement drops the trace event session named my_session:

```
DROP TRACE EVENT SESSION my_session;
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.4.4.159 DROP TRIGGER Statement

Removes a trigger from the database.

Syntax

```
DROP TRIGGER [ IF EXISTS ] [ owner.] [ table-name.]trigger-name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP statement attempts to remove a database object that does not exist.

Privileges

To drop a trigger on a table, one of the following conditions must be true:

- You are the owner of the table.
- You have ALTER privilege on the table.
- You have the ALTER ANY TABLE system privilege.
- You have the ALTER ANY OBJECT system privilege.

To drop a trigger on a view owned by someone else, you must have either the ALTER ANY VIEW or ALTER ANY OBJECT system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

DROP TRIGGER comprises part of optional ANSI/ISO SQL Language Feature T211, "Basic trigger capability". The IF EXISTS clause is not in the standard.

Example

This example creates, and then drops, a trigger called emp_upper_postal_code that ensures postal codes are in upper case before updating the Employees table. If the trigger does not exist, an error is returned.

```
CREATE TRIGGER emp_upper_postal_code
BEFORE UPDATE OF PostalCode
ON GROUPO.Employees
REFERENCING NEW AS new_emp
FOR EACH ROW
WHEN ( ISNUMERIC( new_emp.PostalCode ) = 0 )
BEGIN
   -- Ensure postal code is uppercase (employee might be
   -- in Canada where postal codes contain letters)
   SET new_emp.PostalCode = UPPER(new_emp.PostalCode)
END;
DROP TRIGGER emp_upper_postal_code;
```

Related Information

Dropping a Trigger
CREATE TRIGGER Statement [page 1036]
ALTER TRIGGER Statement [page 768]
ROLLBACK TRIGGER Statement [page 1358]

1.4.4.160 DROP USER Statement

Drops a user.

Syntax

```
DROP USER user-name
```

Parameters

user-name

The name of the user you are dropping.

Remarks

Dropping a user also deletes all database objects (such as tables or procedures) that they own, as well as any external logins for the user. In addition, if the user is specified in the USER clause of any services, then those services are also dropped.

The user being removed cannot be connected to the database when the statement is executed.

If you use this statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY USER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates and then drops the user SQLTester.

```
CREATE USER SQLTester IDENTIFIED BY pass1234;
DROP USER SQLTester;
```

Related Information

Login Policies
ALTER LOGIN POLICY Statement [page 690]
ALTER USER Statement [page 769]
COMMENT Statement [page 808]
CREATE LOGIN POLICY Statement [page 894]
CREATE USER Statement [page 1045]
DROP LOGIN POLICY Statement [page 1101]

1.4.4.161 DROP VARIABLE Statement

Drops a SQL variable.

Syntax

Drop a connection-scope variable:

```
DROP VARIABLE [ IF EXISTS ] identifier
```

Drop a database-scope variable:

```
DROP DATABASE VARIABLE [ IF EXISTS ] [ owner.]identifier
```

Parameters

identifier

A valid identifier for the variable.

owner

Specify the owner of the database-scope variable. If `owner` is not specified, the database server looks for a database-scope variable named `identifier` owned by the user executing the statement. If none is found, the database server looks for a database-scope variable named `identifier` owned by PUBLIC.

IF EXISTS clause

Specify this clause to allow the statement to complete without returning an error if a variable with the specified name (and/or owner, if specified) is not found.

Remarks

The DROP VARIABLE statement drops a SQL variable.

Connection-scope variables are also automatically dropped when the database connection is terminated. Database-scope variables must be explicitly dropped.

If a statement is still accessing a database-scope variable at the time it is dropped, then the variable is still available in memory for that statement only.

Variables are often used for large objects, so dropping them after use or setting them to NULL can free up significant resources such as disk space and memory.

Privileges

Connection-scope variables: No privileges are required to drop a connection-scope variable.

Database-scope variables: No privileges are required to drop a self-owned database-scope variable. To drop a database-scope variable that is owned by another user or by PUBLIC, you must have the MANAGE ANY DATABASE VARIABLE system privilege.

Side Effects

Connection-scope variables: No side effects are associated with dropping a connection-scope variable.

Database-scope variables: Dropping a database-scope variable causes an automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

CREATE VARIABLE Statement [page 1047]
SET Statement [page 1373]

1.4.4.162 DROP VIEW Statement

Removes a view from the database.

Syntax

```
DROP VIEW [ IF EXISTS ] [ owner.]view-name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP VIEW statement attempts to remove a view that does not exist.

When you execute the DROP VIEW statement, the status of all dependent regular views change to INVALID. To determine view dependencies before dropping a view, use the sa_dependent_views system procedure.

If you execute a DROP VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. You must drop the trigger before the view can be dropped or altered.

Privileges

You must be the owner of the view, or have the DROP ANY VIEW or DROP ANY OBJECT system privilege.

Side Effects

Automatic commit. Executing a DROP VIEW statement closes all cursors for the current connection.

When a view is dropped, all procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the fact that the view does not exist. The unloading and loading of procedures and triggers can affect performance if you are regularly dropping and creating views.

Standards

ANSI/ISO SQL Standard

Core Feature. The IF EXISTS clause is not in the standard.

Example

The following example creates a view called MyView, and then drops it. You must be able to select from the Employees table to execute the CREATE VIEW statement in the example.

```
CREATE VIEW MyView
   AS SELECT * FROM GROUPO.Employees;
DROP VIEW MyView;
```

Related Information

CREATE VIEW Statement [page 1051]
ALTER VIEW Statement [page 772]
sa_dependent_views System Procedure [page 1571]

1.4.4.163 EXCEPT Statement

Returns the set difference of two query blocks.

Syntax

```
[ WITH temporary-views ] main-query-block
EXCEPT [ ALL | DISTINCT ] except-query-block
[ ORDER BY [ integer | select-list-expression-name ] [ ASC | DESC ], ... ]
[ FOR XML xml-mode ]
[ OPTION( query-hint, ... ) ]
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| option-name = option-value
```

```
main-query-block :  query-block
```

```
except-query-block : query-block
```

```
option-name : identifier
```

```
option-value :
hostvar (indicator allowed)
| string
| identifier
| number
```

Parameters

main-query-block

A query block, comprising a SELECT statement or a query expression (possibly nested). Query blocks are explained in the documentation for comment elements in SQL.

except-query-block

A query block, comprising a SELECT statement or a query expression (possibly nested). Query blocks are explained in the documentation for comment elements in SQL.

FOR XML clause

This clause is defined in the documentation for the SELECT statement.

OPTION clause

Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- `option-name` = `option-value`.
  Use an `OPTION( isolation_level = ... )` specification in the query text to override all other means of specifying isolation level for a query.

Remarks

The EXCEPT statement returns all rows in main-query-block except those that also appear in the except-query-block. Specify EXCEPT or EXCEPT DISTINCT if you do not want duplicates from main-query-block to appear as duplicates in the result. Otherwise, specify EXCEPT ALL. Query blocks can be nested.

The use of EXCEPT alone is equivalent to EXCEPT DISTINCT.

The `main-query-block` and the `except-query-block` must be UNION-compatible; they must each have the same number of items in their respective SELECT lists, and the types of each expression should be comparable. If corresponding items in two SELECT lists have different data types, the database server chooses a data type for the corresponding column in the result and automatically convert the columns in each `query-block` appropriately.

EXCEPT ALL implements bag difference rather than set difference. For example, if `main-query-block` contains 5 (duplicate) rows with specific values, and `except-query-block` contains 2 duplicate rows with identical values, then EXCEPT ALL will return 3 rows.

The results of EXCEPT are the same as the results of EXCEPT ALL if `main-query-block` does not contain duplicate rows.

The column names displayed are the same column names that are displayed for the first `query-block` and these names are used to determine the expression names to be matched with the ORDER BY clause. An alternative way of customizing result set column names is to use a common table expression (the WITH clause).

Privileges

You must own the tables referenced in `query-block`, or have the SELECT ANY TABLE privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

EXCEPT DISTINCT is a Core Feature. EXCEPT ALL comprises the optional ANSI/ISO SQL Language Feature F304. Explicitly specifying the DISTINCT keyword with EXCEPT is optional ANSI/ISO SQL Language Feature T551. Specifying an ORDER BY clause with EXCEPT is ANSI/ISO SQL Language Feature F850. A `query-block` that contains an ORDER BY clause constitutes ANSI/ISO SQL Feature F851. A query block that contains a row-limit clause (SELECT TOP or LIMIT) comprises optional ANSI/ISO SQL Language Feature F857 or F858, depending on the context. The FOR XML clause and the OPTION clause are not in the standard.

Transact-SQL

EXCEPT is not supported by Adaptive Server Enterprise. However, both EXCEPT ALL and EXCEPT DISTINCT can be used in the Transact-SQL dialect supported by SQL Anywhere.

Related Information

Common Elements in SQL Syntax [page 639]
Set Operators and the NULL Value
SELECT Statement [page 1362]
INTERSECT Statement [page 1242]
UNION Statement [page 1450]
SELECT Statement [page 1362]

1.4.4.164 EXECUTE Statement [ESQL]

Executes a prepared SQL statement.

Syntax

General use

```
EXECUTE statement
[ USING { hostvar-list | [ SQL ] DESCRIPTOR sqlda-name } ]
[ INTO { into-hostvar-list | [ SQL ] DESCRIPTOR into-sqlda-name } ]
[ ARRAY :row-count ]
```

```
row-count : integer | hostvar
```

```
statement :   identifier | hostvar | string
```

```
sqlda-name : identifier
```

```
into-sqlda-name :  identifier
```

Execute immediately

```
EXECUTE IMMEDIATE statement
```

```
statement :  string | hostvar
```

Parameters

USING clause

Results from a SELECT statement or a CALL statement are put into either the variables in the variable list or the program data areas described by the named SQLDA. The correspondence is one-to-one from the OUTPUT (selection list or parameters) to either the host variable list or the SQLDA descriptor array.

INTO clause

If EXECUTE INTO is used with an INSERT statement, the inserted row is returned in the second descriptor. For example, when using auto-increment primary keys or BEFORE INSERT triggers that generate primary key values, the EXECUTE statement provides a mechanism to re-fetch the row immediately and determine the primary key value that was assigned to the row. The same thing can be achieved by using @@identity with auto-increment keys.

ARRAY clause

The optional ARRAY clause can be used with prepared INSERT statements to allow wide inserts, which insert more than one row at a time and which can improve performance. The integer value is the number of rows to be inserted. The SQLDA must contain a variable for each entry (number of rows * number of columns). The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on. Similarly, the ARRAY clause can be used for wide updates, deletes, and merges using prepared UPDATE, DELETE, and MERGE statements.

Remarks

The EXECUTE statement can be used for any SQL statement that can be prepared. Cursors are used for SELECT statements or CALL statements that return many rows from the database.

After successful execution of an INSERT, UPDATE, DELETE, or MERGE statement, the `sqlerrd[2]` field of the SQLCA (SQLCOUNT) is filled in with the number of rows affected by the operation.

Execute

Execute the named dynamic statement, which was previously prepared. If the dynamic statement contains host variable placeholders that supply information for the request (bind variables), either the `sqlda-name` must specify a C variable which is a pointer to a SQLDA containing enough descriptors for all the bind variables occurring in the statement, or the bind variables must be supplied in the `hostvar-list`.

Execute immediately

A short form to PREPARE and EXECUTE a statement that does not contain bind variables or output. The SQL statement contained in the string or host variable is immediately executed, and is dropped on completion.

When performing wide insert, update and delete operations (ARRAY clause), host variables must be simple names and the rows of the wide update must be the same data type.

Privileges

The required privileges depend on the statement being executed.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The EXECUTE statement comprises part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The INTO clause is part of optional ANSI/ISO SQL Language Feature B032, "Extended dynamic SQL". The ARRAY clause is not in the standard.

The EXECUTE IMMEDIATE statement supported with Embedded SQL is also part of optional ANSI/ISO SQL Language Feature B031.

Example

This example executes a DELETE statement.

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM Employees WHERE EmployeeID = 105';
```

This example executes a prepared DELETE statement.

```
EXEC SQL PREPARE del_stmt FROM
'DELETE FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE del_stmt USING :employee_number;
```

This example executes a prepared query.

```
EXEC SQL PREPARE sel1 FROM
'SELECT Surname FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE sel1 USING :employee_number INTO :surname;
```

Related Information

Cursors in Embedded SQL
Named Parameters [page 128]
EXECUTE IMMEDIATE Statement [SP] [page 1155]
PREPARE Statement [ESQL] [page 1308]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]

1.4.4.165 EXECUTE Statement [T-SQL]

Invokes a procedure (an Adaptive Server Enterprise-compatible alternative to the CALL statement), executes a prepared SQL statement in Transact-SQL.

Syntax

Call a stored procedure

```
[ EXECUTE | EXEC ][ @return_status = ] [creator.]procedure_name
[ argument, ... ]
```

```
argument :
[ @parameter-name = ] expression
| [ @parameter-name = ] @variable [ output ]
```

Execute dynamic statements within T-SQL stored procedures and triggers

```
EXECUTE ( string-expression )
```

Remarks

The syntax for calling a stored procedure is implemented for Transact-SQL compatibility. EXECUTE calls a stored procedure, optionally supplying procedure parameters and retrieving output values and return status information. In Watcom SQL, use the CALL or EXECUTE IMMEDIATE statements.

With the syntax for executing statements, you can execute dynamic statements within Transact-SQL stored procedures and triggers. The EXECUTE statement extends the range of statements that can be executed from within procedures and triggers. It lets you execute dynamically prepared statements, such as statements that are constructed using the parameters passed in to a procedure. literal strings in the statement must be enclosed in single quotes, and the statement must be on a single line. This syntax is implemented for Transact-SQL compatibility, but can be used in either Transact-SQL or Watcom SQL batches and procedures.

The Transact-SQL EXECUTE statement does not have a way to signify that a result set is expected. One way to indicate that a Transact-SQL procedure returns a result set is to include something like the following:

```
IF 1 = 0
      SELECT 1 AS a
```

You can also execute statements within Transact-SQL stored procedures and triggers.

Privileges

When calling a procedure, you must be the owner of the procedure, or have the EXECUTE ANY PROCEDURE system privilege.

When executing dynamic statements within T-SQL stored procedures and triggers, the required privileges depend on the statement being executed.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The syntax for calling store procedures is not in the standard.

The syntax for executing dynamic statements offers functionality equivalent to the EXECUTE IMMEDIATE statement in the ANSI/ISO SQL Standard, which is optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". However, the syntax for executing dynamic statements differs from that of the ANSI/ISO SQL Standard.

Example

The following procedure illustrates how to execute a stored procedure.

```
CREATE PROCEDURE p1( @var INTEGER = 54 )
AS
PRINT 'on input @var = %1!', @var
DECLARE @intvar integer
SELECT @intvar=123
SELECT @var=@intvar
PRINT 'on exit @var = %1!', @var;
```

The following statement executes the procedure, supplying the input value of 23 for the parameter. If you are connected from an Open Client or JDBC application, the PRINT messages are displayed in the client window. If you are connected from an ODBC or Embedded SQL application, the messages are displayed in the database server messages window.

```
EXECUTE p1 23;
```

The following is an alternative way of executing the procedure, which is useful if there are several parameters.

```
EXECUTE p1 @var = 23;
```

The following statement executes the procedure, using the default value for the parameter

```
EXECUTE p1;
```

The following statement executes the procedure, and stores the return value in a variable for checking return status.

```
EXECUTE @status = p1 23;
```

Related Information

Named Parameters [page 128]
CALL Statement [page 795]
EXECUTE Statement [ESQL] [page 1151]
EXECUTE IMMEDIATE Statement [SP] [page 1155]

1.4.4.166 EXECUTE IMMEDIATE Statement [SP]

Enables dynamically constructed statements to be executed from within a procedure.

Syntax

```
EXECUTE IMMEDIATE [ execute-option ] string-expression

execute-option :
```

```
WITH QUOTES [ ON | OFF ]
| WITH ESCAPES { ON | OFF }
| WITH BATCH { ON | OFF }
| WITH RESULT SET { ON | OFF }
```

Parameters

WITH QUOTES clause

When you specify WITH QUOTES or WITH QUOTES ON, any double quotes in the string-expression are assumed to delimit an identifier. When you do not specify WITH QUOTES, or specify WITH QUOTES OFF, the treatment of double quotes in the string-expression depends on the current setting of the quoted_identifier option.

WITH QUOTES is useful when an object name that is passed into the stored procedure is used to construct the statement that is to be executed, but the name might require double quotes and the procedure might be called when the quoted_identifier option is set to Off.

WITH ESCAPES clause

WITH ESCAPES OFF causes any escape sequences (such as \n, \x, or \\) in the string-expression to be ignored. For example, two consecutive backslashes remain as two backslashes, rather than being converted to a single backslash. The default setting is equivalent to WITH ESCAPES ON.

One use of WITH ESCAPES OFF is for easier execution of dynamically constructed statements referencing file names that contain backslashes.

In some contexts, escape sequences in the `string-expression` are transformed before the EXECUTE IMMEDIATE statement is executed. For example, compound statements are parsed before being executed, and escape sequences are transformed during this parsing, regardless of the WITH ESCAPES setting. In these contexts, WITH ESCAPES OFF prevents further translations from occurring. For example:

```
BEGIN
   DECLARE String1 LONG VARCHAR;
   DECLARE String2 LONG VARCHAR;
   EXECUTE IMMEDIATE
        'SET String1 = ''One backslash: \\\\ ''';
     EXECUTE IMMEDIATE WITH ESCAPES OFF
        'SET String2 = ''Two backslashes: \\\\ ''';
   SELECT String1, String2
END
```

WITH BATCH clause

The WITH BATCH clause allows you to control the execution of batches in EXECUTE IMMEDIATE statements. Setting WITH BATCH OFF provides protection against inadvertent SQL-injection when the procedure is run.

WITH BATCH ON is the default, except for procedures owned by dbo.

When WITH BATCH OFF is used, the statement specified by `string-expression` must be a single statement.

WITH RESULT SET clause

The WITH RESULT SET clause allows the server to define correctly the procedure containing it. Specifying WITH RESULT SET ON or WITH RESULT SET OFF affects both what happens when the procedure is

created, as well as what happens when the procedure is executed. The default option is WITH RESULT SET OFF.

You can have an EXECUTE IMMEDIATE statement return a result set by specifying WITH RESULT SET ON. With this clause, the containing procedure is marked as returning a result set.

Remarks

The EXECUTE IMMEDIATE statement extends the range of statements that can be executed from within procedures and triggers. It lets you execute dynamically prepared statements, such as statements that are constructed using the parameters passed in to a procedure.

Literal strings in the statement must be enclosed in single quotes. String literals cannot span multiple lines.

Only global variables can be referenced in a statement executed by EXECUTE IMMEDIATE.

Statements executed with EXECUTE IMMEDIATE do not have their plans cached.

Privileges

None.

Side Effects

None. However, if the statement is a data definition statement with an automatic commit as a side effect, that commit does take place.

Standards

ANSI/ISO SQL Standard

EXECUTE IMMEDIATE is optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The `execute-option` syntax is not in the standard. The ANSI/ISO SQL Standard prohibits the use of EXECUTE IMMEDIATE that returns a result set.

Example

The following procedure creates a table, where the table name is supplied as a parameter to the procedure.

```
CREATE PROCEDURE CreateTableProc(
                  IN tablename char(30)
```

```
                    )
BEGIN
   EXECUTE IMMEDIATE
   'CREATE TABLE ' || tablename ||
   ' ( column1 INT PRIMARY KEY)'
END;
```

To call the procedure and create a table called mytable:

```
CALL CreateTableProc( 'mytable' );
```

Related Information

EXECUTE IMMEDIATE Used in Procedures, Triggers, User-defined Functions, and Batches
Named Parameters [page 128]
quoted_identifier Option
CREATE PROCEDURE Statement [page 913]
CREATE PROCEDURE Statement [External Call] [page 921]
CREATE PROCEDURE Statement [Web Service] [page 931]
BEGIN Statement [page 784]
EXECUTE Statement [ESQL] [page 1151]
EXECUTE Statement [T-SQL] [page 1153]

1.4.4.167 EXIT Statement [Interactive SQL]

Leaves Interactive SQL.

Syntax

```
{ EXIT | QUIT | BYE } [ return-code ]
```

```
return-code : number | connection-variable
```

Remarks

This statement closes the Interactive SQL window if you are running Interactive SQL as a windowed program, or terminates Interactive SQL altogether when running in command-prompt (batch) mode. In both cases, the database connection is also closed. Before closing the database connection, Interactive SQL automatically executes a COMMIT statement if the commit_on_exit option is set to On. If this option is set to Off, Interactive SQL performs an implicit ROLLBACK. By default, the commit_on_exit option is set to On.

The optional return code can be checked in batch files to determine the success or failure of the statements in an Interactive SQL script file. The default return code is 0.

Privileges

None.

Side Effects

This statement automatically performs a commit if option commit_on_exit is set to On (the default); otherwise it performs an implicit rollback.

On Windows operating systems the optional return value is available as ERRORLEVEL.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example sets the Interactive SQL return value to 1 if there are any rows in table T, or to 0 if T contains no rows.

```
CREATE VARIABLE rowCount INT;
CREATE VARIABLE retcode INT;
SELECT COUNT(*) INTO rowCount FROM GROUPO.Products;
IF( rowCount > 0 ) THEN
    SET retcode = 1;
ELSE
    SET retcode = 0;
END IF;
EXIT retcode;
```

> **i** Note
>
> You cannot write the following statement because EXIT is an Interactive SQL statement (not a SQL statement), and you cannot include any Interactive SQL statement in other SQL block statements.
>
> ```
> CREATE VARIABLE rowCount INT;
> SELECT COUNT(*) INTO rowCount FROM T;
> IF( rowCount > 0 ) THEN
>     EXIT 1    //  <-- not allowed
> ELSE
>     EXIT 0    //  <-- not allowed
> END IF;
> ```

Related Information

Interactive SQL
SET OPTION Statement [page 1387]

1.4.4.168 EXPLAIN Statement [ESQL]

Retrieves a text specification of the optimization strategy used for a particular cursor.

Syntax

```
EXPLAIN PLAN FOR CURSOR cursor-name
{  INTO hostvar | USING DESCRIPTOR sqlda-name }
```

```
cursor-name : identifier | hostvar
```

```
sqlda-name : identifier
```

Remarks

The EXPLAIN statement retrieves a text representation of the optimization strategy for the named cursor. The cursor must be previously declared and opened.

The `hostvar` or `sqlda-name` variable must be of string type. The optimization string specifies in what order the tables are searched, and also which indexes are being used for the searches if any.

This string may be long, depending on the query, and has the following format:

```
table (index), table (index), ...
```

If a table has been given a correlation name, the correlation name will appear instead of the table name. The order that the table names appear in the list is the order in which they are accessed by the database server. After each table is a parenthesized index name. This is the index that is used to access the table. If no index is used (the table is scanned sequentially) the letters "seq" will appear for the index name. If a particular SQL SELECT statement involves subqueries, a colon (:) will separate each subquery's optimization string. These subquery sections will appear in the order that the database server executes the queries.

After successful execution of the EXPLAIN statement, the sqlerrd field of the SQLCA (SQLIOESTIMATE) is filled in with an estimate of the number of input/output operations required to fetch all rows of the query.

You can only execute this statement on cursors you opened.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example illustrates the use of EXPLAIN:

```
EXEC SQL BEGIN DECLARE SECTION;
char plan[300];
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE employee_cursor CURSOR FOR
   SELECT EmployeeID, Surname
   FROM Employees
   WHERE Surname like :pattern;
EXEC SQL OPEN employee_cursor;
EXEC SQL EXPLAIN PLAN FOR CURSOR employee_cursor INTO :plan;
printf( "Optimization Strategy: '%s'.n", plan );
```

The plan variable contains the following string:

```
'Employees <seq>'
```

Related Information

Cursors in Embedded SQL
The SQL Communication Area (SQLCA)
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
PREPARE Statement [ESQL] [page 1308]
FETCH Statement [ESQL] [SP] [page 1162]
CLOSE statement [ESQL] [SP] [page 807]
OPEN Statement [ESQL] [SP] [page 1286]

1.4.4.169 FETCH Statement [ESQL] [SP]

Positions, or re-positions, a cursor to a specific row, and then copies expression values from that row into variables accessible from within the stored procedure or application.

Syntax

Stored procedures

```
FETCH [ cursor-position ] cursor-name
INTO variable-list [ FOR UPDATE ]
```

Embedded SQL

```
FETCH [ cursor-position ] cursor-name
[ INTO { hostvar-list } | USING [ SQL ] DESCRIPTOR sqlda-name ]
[ PURGE ]
[ BLOCK n ]
[ FOR UPDATE ]
[ ARRAY fetch-count ]
```

```
cursor-position :
    NEXT | PRIOR | FIRST | LAST
| { ABSOLUTE | RELATIVE } row-count
```

```
row-count : number | hostvar
```

```
cursor-name : identifier | hostvar
```

```
hostvar-list : may contain indicator variables
```

```
variable-list :  stored procedure variables
```

```
sqlda-name :  identifier
```

```
fetch-count :  integer | hostvar
```

Parameters

INTO clause

The INTO clause is optional. If it is not specified, the FETCH statement positions the cursor only. The `hostvar-list` is for Embedded SQL use only.

> **Note**
>
> If `variable-list` only contains one item, and there are multiple items in the result set of the cursor, then `variable-list` is considered a row variable and the returned column values are assigned to the fields of the row variable in order.

cursor position

An optional positional parameter allows the cursor to be moved before a row is fetched. If not specified, NEXT is assumed. If the fetch includes a positioning parameter and the position is outside the allowable cursor positions, the SQLE_NOTFOUND warning is issued and the SQLCOUNT field indicates the offset from a valid position.

The OPEN statement initially positions the cursor before the first row.

NEXT clause

Next is the default positioning, and causes the cursor to advance one row before the row is fetched.

PRIOR clause

Causes the cursor to back up one row before fetching.

RELATIVE clause

RELATIVE positioning is used to move the cursor by a specified number of rows in either direction before fetching. A positive number indicates moving forward and a negative number indicates moving backward. So, a NEXT is equivalent to RELATIVE 1 and PRIOR is equivalent to RELATIVE -1. RELATIVE 0 retrieves the same row as the last fetch statement on this cursor.

ABSOLUTE clause

The ABSOLUTE positioning parameter is used to go to a particular row. A zero indicates the position before the first row.

A one (1) indicates the first row, and so on. Negative numbers are used to specify an absolute position from the end of the cursor. A negative one (-1) indicates the last row of the cursor.

FIRST clause

A short form for ABSOLUTE 1.

LAST clause

A short form for ABSOLUTE -1.

> **i** Note
>
> Inserts and some updates to DYNAMIC SCROLL cursors can cause problems with cursor positioning. The database server does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear until the cursor is closed and opened again.
>
> This behavior occurs if a temporary table had to be created to open the cursor.
>
> The UPDATE statement can cause a row to move in the cursor. This will happen if the cursor has an ORDER BY that uses an existing index (a temporary table is not created).

BLOCK clause

Rows may be fetched by the client application more than one at a time. This is referred to as block fetching, prefetching, or multi-row fetching. The first fetch causes several rows to be sent back from the database server. The client buffers these rows, and subsequent fetches are retrieved from these buffers without a new request to the database server.

The BLOCK clause is for use in Embedded SQL only. It gives the client and server a hint about how many rows may be fetched by the application. The special value of 0 means the request is sent to the database server and a single row is returned (no row blocking). The BLOCK clause will reduce the number of rows

included in the next prefetch to the BLOCK value. To increase the number of rows prefetched, use the PrefetchRows connection parameter.

If you do not specify a BLOCK clause, the value specified on OPEN is used.

FETCH RELATIVE 0 always re-fetches the row.

If prefetch is disabled for the cursor, the BLOCK clause is ignored and rows are fetched one at a time. If ARRAY is also specified, then the number of rows specified by ARRAY are fetched.

PURGE clause

The PURGE clause is for use in Embedded SQL only. It causes the client to flush its buffers of all rows, and then send the fetch request to the database server. This fetch request may return a block of rows.

FOR UPDATE clause

The FOR UPDATE clause indicates that the fetched row will subsequently be updated with an UPDATE WHERE CURRENT OF CURSOR statement. This clause causes the database server to put an intent lock on the row. The lock is held until the end of the current transaction.

ARRAY clause

The ARRAY clause is for use in Embedded SQL only. It allows so-called wide fetches, which retrieve more than one row at a time, and which may improve performance.

To use wide fetches in Embedded SQL, include the fetch statement in your code as follows:

```
EXEC SQL FETCH ... ARRAY nnn
```

where ARRAY `nnn` is the last item of the FETCH statement. The fetch count `nnn` can be a host variable. The SQLDA must contain nnn * (columns per row) variables. The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.

Remarks

The FETCH statement retrieves one row from the named cursor. The cursor must have been previously opened.

Table reference variables (variables defined as type TABLE REF) are not supported for use in FETCH statements.

Embedded SQL use

The Embedded SQL FETCH statement does not support arrays.

A DECLARE CURSOR statement must appear before the FETCH statement in the C source code, and the OPEN statement must be executed before the FETCH statement. If a host variable is being used for the cursor name, the DECLARE statement actually generates code and must be executed before the FETCH statement.

The server returns in SQLCOUNT the number of records fetched, and always returns a SQLCOUNT greater than zero unless there is an error or warning.

If the SQLSTATE_NOTFOUND warning is returned on the fetch, the `sqlerrd[2]` field of the SQLCA (SQLCOUNT) contains the number of rows by which the attempted fetch exceeded the allowable cursor positions. The value is 0 if the row was not found but the position is valid; for example, executing FETCH

RELATIVE 1 when positioned on the last row of a cursor. The value is positive if the attempted fetch was beyond the end of the cursor, and negative if the attempted fetch was before the beginning of the cursor. The cursor is positioned on the last row if the attempted fetch was beyond the end of the cursor, and on the first row if the attempted fetch was before the beginning of the cursor.

After successful execution of the fetch statement, the `sqlerrd[1]` field of the SQLCA (SQLIOCOUNT) is incremented by the number of input/output operations required to perform the fetch. This field is actually incremented on every database statement.

Single row fetch

One row from the result of the SELECT statement is put into the variables in the variable list. The correspondence is one-to-one from the SELECT list to the host variable list.

Multi-row fetch

One or more rows from the result of the SELECT statement are put into either the variables in `variable-list` or the program data areas described by `sqlda-name`. In either case, the correspondence is one-to-one from the SELECT list to either the `hostvar-list` or the `sqlda-name` descriptor array.

Privileges

The cursor must be opened and you must have the SELECT object-level privilege on the tables, or be owner of the tables referenced in the declaration of the cursor, or have the SELECT ANY TABLE system privilege.

Side Effects

A FETCH statement may cause multiple rows to be retrieved from the server to the client if prefetching is enabled.

Standards

ANSI/ISO SQL Standard

With minor exceptions, the stored procedure syntax of the FETCH statement is a Core Feature of the ANSI/ISO SQL Standard. Scrolling options other than NEXT constitute optional ANSI/ISO SQL Language Feature F431, "Read-only scrollable cursors". The software does not support the optional FROM clause of the FETCH statement as documented in the Standard.

The Embedded SQL syntax is not in the standard.

The FOR UPDATE, PURGE, ARRAY, BLOCK, and USING [SQL] DESCRIPTOR clauses are not in the standard.

Example

The following is an Embedded SQL example:

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT EmployeeID, Surname FROM Employees;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

The following is a procedure example:

```
BEGIN
   DECLARE cur_employee CURSOR FOR
      SELECT Surname
      FROM Employees;
   DECLARE name CHAR(40);
   OPEN cur_employee;
   lp: LOOP
      FETCH NEXT cur_employee into name;
      IF SQLCODE <> 0 THEN LEAVE lp END IF;
       ...
   END LOOP;
   CLOSE cur_employee;
END
```

Related Information

How Locking Works
How to Fetch Data Using Embedded SQL
Wide Fetches Using Embedded SQL
Cursors in Embedded SQL
Cursors in Procedures, Triggers, User-defined Functions, and Batches
SELECT Statement [page 1362]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
FOR Statement [page 1166]
PREPARE Statement [ESQL] [page 1308]
OPEN Statement [ESQL] [SP] [page 1286]
RESUME Statement [page 1341]

1.4.4.170 FOR Statement

Repeats the execution of a statement list once for each row in a cursor.

Syntax

```
[ statement-label : ]
FOR for-loop-name AS cursor-name [ cursor-type ] CURSOR
 { FOR statement [ FOR { UPDATE [ cursor-concurrency ] | READ ONLY } ]
```

```
    | USING variable-name }
DO statement-list
END FOR [ statement-label ]

cursor-type :
NO SCROLL
| DYNAMIC SCROLL
| SCROLL
| INSENSITIVE
| SENSITIVE

cursor-concurrency : BY { VALUES | TIMESTAMP | LOCK }

variable-name : identifier
```

Parameters

NO SCROLL clause

A cursor declared NO SCROLL is restricted to moving forward through the result set using FETCH NEXT and FETCH RELATIVE 0 seek operations.

As rows cannot be returned to once the cursor leaves the row, there are no sensitivity restrictions on the cursor. When a NO SCROLL cursor is requested, the database server supplies the most efficient kind of cursor, which is an asensitive cursor.

DYNAMIC SCROLL clause

DYNAMIC SCROLL is the default cursor type. DYNAMIC SCROLL cursors can use all formats of the FETCH statement.

When a DYNAMIC SCROLL cursor is requested, the database server supplies an asensitive cursor. When using cursors there is always a trade-off between efficiency and consistency. Asensitive cursors provide efficient performance at the expense of consistency.

SCROLL clause

A cursor declared SCROLL can use all formats of the FETCH statement. When a SCROLL cursor is requested, the database server supplies a value-sensitive cursor.

The database server must execute value-sensitive cursors in such a way that result set membership is guaranteed. DYNAMIC SCROLL cursors are more efficient and should be used unless the consistent behavior of SCROLL cursors is required.

INSENSITIVE clause

A cursor declared INSENSITIVE has its values and membership fixed over its lifetime. The result set of the SELECT statement is materialized when the cursor is opened. FETCHING from an INSENSITIVE cursor does not see the effect of any other INSERT, UPDATE, MERGE, PUT, or DELETE statement from any connection, including the connection that opened the cursor.

SENSITIVE clause

A cursor declared SENSITIVE is sensitive to changes to membership or values of the result set.

FOR UPDATE clause

FOR UPDATE is the default. Cursors default to FOR UPDATE for single-table queries without an ORDER BY clause, or if the ansi_update_constraints option is set to Off. When the ansi_update_constraints option is set to Cursors or Strict, then cursors over a query containing an ORDER BY clause default to READ ONLY. However, you can explicitly mark cursors as updatable using the FOR UPDATE clause.

FOR READ ONLY clause

A cursor declared FOR READ ONLY cannot be used in UPDATE (positioned), DELETE (positioned), or PUT statements. Because it is expensive to allow updates over cursors with an ORDER BY clause or a join, cursors over a query containing a join of two or more tables are READ ONLY and cannot be made updatable unless the ansi_update_constraints database option is Off. In response to any request for a cursor that specifies FOR UPDATE, the database server provides either a value-sensitive cursor or a sensitive cursor. Insensitive and asensitive cursors are not updatable.

Remarks

The FOR statement is a control statement that allows you to execute a list of SQL statements once for each row in a cursor. The FOR statement is equivalent to a compound statement with a DECLARE for the cursor and a DECLARE of a variable for each column in the result set of the cursor followed by a loop that fetches one row from the cursor into the local variables and executes `statement-list` once for each row in the cursor.

Valid cursor types include dynamic scroll (default), scroll, no scroll, sensitive, and insensitive.

The name and data type of each local variable is derived from the `statement` used in the cursor. With a SELECT statement, the data types are the data types of the expressions in the SELECT list. The names are the SELECT list item aliases, if they exist; otherwise, they are the names of the columns. Any SELECT list item that is not a simple column reference must have an alias. With a CALL statement, the names and data types are taken from the RESULT clause in the procedure definition.

The LEAVE statement can be used to resume execution at the first statement after the END FOR. If the ending `statement-label` is specified, it must match the beginning `statement-label`.

The cursor created by a FOR statement is implicitly opened WITH HOLD, so statements executed within the loop that cause a COMMIT do not cause the cursor to be closed.

> ⚠ Caution
>
> If you do not specify `cursor-name`, `cursor-type` is used as `cursor-name`.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The FOR statement is part of optional ANSI/ISO SQL Language Feature P002, "Computational completeness". The USING clause of the FOR statement is not in the standard. As with the DECLARE CURSOR statement, the use of `cursor-concurrency` is not in the standard, nor are the combinations of cursor sensitivity and cursor scrollability options.

Example

The following fragment illustrates the use of the FOR loop:

```
FOR names AS curs INSENSITIVE CURSOR FOR
SELECT Surname
FROM Employees
DO
   CALL search_for_name( Surname );
END FOR;
```

This fragment also illustrates the use of the FOR loop:

```
BEGIN
    FOR names AS curs SCROLL CURSOR FOR
    SELECT EmployeeID, GivenName FROM Employees where EmployeeID < 130
    FOR UPDATE BY VALUES
    DO
        MESSAGE 'emp: ' || GivenName;
    END FOR;
END
```

The following example shows the FOR loop being using inside of a procedure called myproc, which returns the top 10 employees from the Employees table, depending on the sort order specified when calling the procedure (asc for ascending, and desc for descending):

```
CALL sa_make_object( 'procedure', 'myproc' ) ;
ALTER PROCEDURE myproc (
    IN @order_by VARCHAR(20) DEFAULT NULL
)
RESULT ( Surname person_name_t )
BEGIN
    DECLARE @sql LONG VARCHAR;
    DECLARE @msg LONG VARCHAR;
    DECLARE LOCAL TEMPORARY TABLE temp_names( surnames person_name_t );
    SET @sql = 'SELECT TOP(10) * FROM Employees AS t ' ;
    CASE @order_by
    WHEN 'asc' THEN
        SET @sql = @sql || 'ORDER BY t.Surname ASC';
        SET @msg = 'Sorted ascending by last name: ';
    WHEN 'desc' THEN
        SET @sql = @sql || 'ORDER BY t.Surname DESC';
        SET @msg = 'Sorted ascending by last name: ';
    END CASE;
    FOR loop_name AS curs SCROLL CURSOR USING @sql
    DO
        INSERT INTO temp_names( surnames ) VALUES( Surname );
        MESSAGE( @msg || Surname ) ;
    END FOR;
```

```
    SELECT * FROM temp_names;
END ;
```

Calling the myproc procedure and specifying asc (for example, `CALL myproc( 'asc' );`) returns the following results:

Surname
Ahmed
Barker
Barletta
Bertrand
Bigelow
Blaikie
Braun
Breault
Bucceri
Butterfield

Related Information

Sensitive Cursors
Insensitive Cursors
Value-sensitive Cursors
Asensitive Cursors
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
FETCH Statement [ESQL] [SP] [page 1162]
CONTINUE Statement [page 818]
LOOP Statement [page 1269]

1.4.4.171 FORWARD TO Statement

Sends native syntax SQL statements to a remote server.

Syntax

Forward a single statement

```
FORWARD TO server-name sql-statement
```

Enter passthrough mode

```
FORWARD TO [ server-name ]
```

Exit passthrough mode

FORWARD TO

Parameters

server-name

The name of the remote server.

sql-statement

A command in the native SQL syntax of the remote server. The command or group of commands is enclosed in braces ({}) or single quotes.

Remarks

The FORWARD TO statement enables users to specify the server to which a passthrough connection is required. The statement can be used in two ways:

Syntax for forwarding a single statement

Send a single statement to a remote server.

Syntax for passthrough mode

Place the database server into passthrough mode for sending a series of statements to a remote server. All subsequent statements are passed directly to the remote server. To turn passthrough mode off, execute FORWARD TO without a `server-name` specification.

If you encounter an error from the remote server while in passthrough mode, you must still execute a FORWARD TO statement to turn passthrough off.

Proxy Tables: The user must login to connect to a remote server using this syntax. If the FORWARD TO statement appears in a procedure owned by a different owner, or if the statement appears in an event (or any procedure called by an event), no authentication has taken place and hence there is no password to be forwarded. The only way that proxy table usage can work in this scenario is to define an EXTERNLOGIN.

When establishing a connection to server-name on behalf of the user, the database server uses one of the following:

- A remote login alias set using CREATE EXTERNLOGIN
- If a remote login alias is not set up, the name and password used to communicate with the database server

If the connection cannot be made to the server specified, the reason is contained in a message returned to the user.

After statements are passed to the requested server, any results are converted into a form that can be recognized by the client program.

> **Note**
>
> The FORWARD TO statement is a server directive and cannot be used in stored procedures, triggers, events, or batches. However, the dbo.sp_forward_to_remote_server() system procedure can be used in stored procedures, triggers, events or batches.

Privileges

None.

Side Effects

The remote connection is set to AUTOCOMMIT (unchained) mode for the duration of the FORWARD TO session. Any work that was pending before the FORWARD TO statement is automatically committed.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example sends a SQL statement to the remote server RemoteASE:

```
FORWARD TO RemoteASE { SELECT * FROM titles };
```

The following example shows a passthrough session with the remote server aseprod:

```
FORWARD TO aseprod;
    SELECT * FROM titles;
    SELECT * FROM authors;
FORWARD TO;
```

Related Information

PASSTHROUGH Statement [SQL Remote] [page 1299]
sp_forward_to_remote_server System Procedure [page 1776]

1.4.4.172 FROM Clause

Specifies the database tables or views involved in a DELETE, SELECT, or UPDATE statement. When used within a SELECT statement, the FROM clause can also be used in a MERGE or INSERT statement.

Syntax

```
FROM table-expression,...
```

```
table-expression :
table-name
| view-name
| procedure-name
| derived-table
| lateral-derived-table
| pivoted-derived-table
| unpivoted-derived-table
| join-expression
| ( table-expression,... )
| openstring-expression
| apply-expression
| contains-expression
| dml-derived-table
| openxml-operator
| array-operator
```

```
table-name :
[ userid.]table-name
[ [ AS ] correlation-name ]
[ WITH ( hint [...] ) ]
[ FORCE INDEX ( index-name ) ]
```

```
view-name :
[ userid.]view-name [ [ AS ] correlation-name ]
[ WITH ( table-hint ) ]
```

```
procedure-name :
[ owner.]procedure-name ( [ parameter,... ] )
[ WITH ( column-name data-type,... ) ]
[ [ AS ] correlation-name ]
```

```
derived-table :
( select-statement )
[ AS ] correlation-name [ ( column-name,... ) ]
```

```
lateral-derived-table :
LATERAL ( select-statement | table-expression )
[ AS ] correlation-name [ ( column-name,... ) ]
```

```
pivoted-derived-table :
pivot-source-table PIVOT [ XML ] ( pivot-clause ) [ AS ] pivoted-correlation-
name
```

```
unpivoted-derived-table :
unpivoted-derived-table :
unpivot-source-table UNPIVOT [ { INCLUDE | EXCLUDE } NULLS ]  ( unpivot-
clause ) [ AS ] correlation-name
```

```
join-expression :
table-expression join-operator table-expression
[ ON join-condition ]
```

```
join-operator :
[ KEY | NATURAL ] [ join-type ] JOIN
| CROSS JOIN
```

```
join-type :
INNER
| LEFT [ OUTER ]
| RIGHT [ OUTER ]
| FULL [ OUTER ]
```

```
hint :
table-hint | index-hint
```

```
table-hint :
READPAST
| UPDLOCK
| XLOCK
| FASTFIRSTROW
| HOLDLOCK
| NOLOCK
| READCOMMITTED
| READUNCOMMITTED
| REPEATABLEREAD
| SERIALIZABLE
```

```
index-hint :
NO INDEX
| INDEX ( [ PRIMARY KEY | FOREIGN KEY ] index-name [, ..] ) [ INDEX ONLY { ON |
OFF } ]
| CLUSTERED INDEX [ INDEX ONLY { ON | OFF } ]
```

```
openstring-expression :
OPENSTRING ( { FILE | VALUE } string-expression )
WITH ( rowset-schema )
 [ OPTION ( scan-option ... ) ]
 [ AS ] correlation-name
```

```
apply-expression :
table-expression { CROSS | OUTER } APPLY table-expression
```

```
contains-expression :
{ table-name | view-name } CONTAINS ( column-name [,...], contains-query )
[ [ AS ] score-correlation-name ]
```

```
rowset-schema :
column-schema-list
| TABLE [owner.]table-name [ ( column-list ) ]
```

```
column-schema-list :
{ column-name user-or-base-type |  filler( ) } [ ,... ]
```

```
column-list :
{ column-name | filler( ) } [ ,... ]
```

```
scan-option :
BYTE ORDER MARK { ON | OFF }
| COMMENTS INTRODUCED BY comment-prefix
| { COMPRESSED | AUTO | NOT COMPRESSED }
| DELIMITED BY string
| ENCODING encoding
| { ENCRYPTED KEY key-expression | NOT ENCRYPTED }
| ESCAPE CHARACTER character
| ESCAPES { ON | OFF }
| FORMAT { TEXT | BCP }
| HEXADECIMAL { ON | OFF }
| QUOTE string
| QUOTES { ON | OFF }
| ROW DELIMITED BY string
| SKIP integer
| STRIP { ON | OFF | LTRIM | RTRIM | BOTH }
```

```
key-expression : string | variable
```

```
contains-query : string
```

```
dml-derived-table :
( dml-statement ) REFERENCING ( [ table-version-names | NONE ] )
```

```
dml-statement :
insert-statement
delete-statement
update-statement
merge-statement
```

```
table-version-names :
OLD [ AS ] correlation-name [ FINAL [ AS ] correlation-name ]
| FINAL [ AS ] correlation-name
```

Parameters

table-name

A base table or temporary table. Tables owned by a different user can be qualified by specifying the user ID. Tables owned by user-defined roles that the user is a grantee of are found by default without specifying the user ID.

view-name

Specifies a view to include in the query. As with tables, views owned by a different user can be qualified by specifying the user ID. Views owned by groups to which the current user belongs are found by default without specifying the user ID. Although the syntax permits table hints on views, these hints have no effect.

procedure-name

A stored procedure that returns a result set. This clause applies to the FROM clause of SELECT statements only. The parentheses following the procedure name are required even if the procedure does not take parameters. DEFAULT can be specified in place of an optional parameter.

The argument list can be specified by position or by using keyword format. By position, the arguments match up with the corresponding parameter in the parameter list for the procedure. By keyword, the arguments are matched up with the named parameters.

If the stored procedure returns multiple result sets, only the first one is used.

The WITH clause provides a way of specifying column name aliases for the procedure result set. If a WITH clause is specified, the number of columns must match the number of columns in the procedure result set, and the data types must be compatible with those in the procedure result set. If no WITH clause is specified, the column names and types are those defined by the procedure definition.

For Embedded SQL applications, when you create a procedure without a RESULT clause and the procedure returns a variable result set, a DESCRIBE of the SELECT statement referencing the procedure may fail. To prevent the failure of the DESCRIBE, include a WITH clause that describes the expected result set schema.

Selecting from a procedure generates a temporary table. For example, the following query creates a local temporary table:

```
BEGIN
...
    my: LOOP
      BEGIN
        SELECT TOP 1 NUMBER INTO conn_id FROM sa_conn_info( ) ORDER BY NUMBER
DESC;
      END;
...
   END LOOP my;
END
```

derived-table

You can supply a SELECT statement instead of table or view name in the FROM clause. A SELECT statement used in this way is called a derived table, and it must be given an alias. For example, the following statement contains a derived table, MyDerivedTable, which ranks products in the Products table by UnitPrice.

```
SELECT TOP 3 *
       FROM ( SELECT Description,
                     Quantity,
                     UnitPrice,
                     RANK() OVER ( ORDER BY UnitPrice ASC )
                 AS Rank
                 FROM GROUPO.Products ) AS MyDerivedTable
ORDER BY Rank;
```

lateral-derived-table

A derived table, stored procedure, or joined table that may include references to objects in the parent statement (outer references). Use a lateral derived table to use an outer reference in the FROM clause.

Use outer references only to tables that precede the lateral derived table in the FROM clause. For example, you cannot use an outer reference to an item in the SELECT list.

The table and the outer reference must be separated by a comma. For example, the following queries are valid:

```
SELECT *
 FROM A, LATERAL( B LEFT OUTER JOIN C ON ( A.x = B.x ) ) myLateralDT;
```

```
SELECT *
 FROM A, LATERAL( SELECT * FROM B WHERE A.x = B.x ) myLateralDT;
```

```
SELECT *
 FROM A, LATERAL( procedure-name( A.x ) ) myLateralDT;
```

Specifying LATERAL (`table-expression`) is equivalent to specifying LATERAL (SELECT * FROM `table-expression`).

pivoted-derived-table Pivots data in the FROM `table-name` expression into a pivoted derived table. The full syntax for this clause is described in the PIVOT clause topic.

unpivoted-derived-table Unpivots data in the FROM `table-name` expression into an unpivoted derived table. The full syntax for this clause is described in the UNPIVOT clause topic.

openstring-expression

Specify an OPENSTRING clause to query within a file or a BLOB, treating the content of these sources as a set of rows. When doing so, you also specify information about the schema of the file or BLOB for the result set to be generated since you are not querying a defined structure such as a table or view. This clause applies to the FROM clause of a SELECT statement. It is not supported for UPDATE or DELETE statements.

The ROWID function is supported over the result set of a table generated by an OPENSTRING expression.

The following subclauses and parameters of the OPENSTRING clause are used to define and query data within files and BLOBs:

FILE and VALUE clauses

Use the FILE clause to specify the file to query. Use the VALUE clause to specify the BLOB expression to query. The data type for the BLOB expression is assumed to be LONG BINARY. You can specify the READ_CLIENT_FILE function as a value to the VALUE clause.

If neither the FILE nor VALUE keyword is specified, VALUE is assumed.

When using FORMAT SHAPEFILE, only FILE is assumed.

WITH clause

Use this clause to specify the rowset schema (column names and data types) of the data being queried. You can specify the columns directly (for example, `WITH ( Surname CHAR(30), GivenName CHAR(30) )`). You can also use the TABLE subclause to reference a table to use to obtain schema information from (for example, `WITH ( TABLE dba.Employees ( Surname, GivenName ) )`). You must own or have SELECT privileges on the table you specify.

When specifying columns, you can specify filler( ) for columns that you want to skip in the input data (for example, `WITH ( filler( ), Surname CHAR(30), GivenName CHAR(30) )`).

OPTION clause

Use the OPTION clause to specify parsing options to use for the input file, such as escape characters, delimiters, encoding, and so on. Supported options comprise those options for the LOAD TABLE statement that control the parsing of an input file.

scan-option

For information about each scan option, see the load options for the LOAD TABLE statement.

apply-expression

Use this clause to specify a join condition where the right `table-expression` is evaluated for every row in the left `table-expression`. For example, you can use an apply expression to evaluate a function, procedure, or derived table for each row in a table expression.

contains-expression

Use the CONTAINS clause following a table name to filter the table and return only those rows matching the full text query specified with `contains-query`. Every matching row of the table is returned together with a score column that can be referred to by using `score-correlation-name`. If `score-correlation-name` is not specified, then the score column can be referred to by the default correlation name, contains.

With the exception of the optional correlation name argument, the CONTAINS clause takes the same arguments as the CONTAINS search condition.

There must be a text index on the columns listed in the CONTAINS clause.

The `contains-query` cannot be NULL or an empty string. If the text configuration settings cause all of the terms in the `contains-query` to be dropped, rows from the base table referenced by the `contains-expression` are not returned.

correlation-name

Use `correlation-name` to specify a substitute name for a table or view in the FROM clause. The substitute name can then be referenced from elsewhere in the statement. For example, emp and dep are correlation names for the Employees and Departments tables, respectively:

```
SELECT Surname, GivenName, DepartmentName
   FROM GROUPO.Employees emp, GROUPO.Departments dep,
   WHERE emp.DepartmentID=dep.DepartmentID;
```

Correlation names can be used to distinguish between different instances of the same table. For example, the following query joins the Employee table to itself, using the Mgr correlation name, to include the surname of each employee's manager in the result:

```
SELECT Emp.EmployeeID, Emp.Surname, Dept.DepartmentName, Mgr.Surname AS
ManagerName
    FROM GROUPO.Employees AS Emp, GROUPO.Departments AS Dept,
GROUPO.Employees AS Mgr
    WHERE Emp.DepartmentID = Dept.DepartmentID AND Emp.ManagerID =
Mgr.EmployeeID;
```

dml-statement

Use `dml-statement` to specify the DML statement (INSERT, DELETE, UPDATE, or MERGE) from which you want to select rows. During execution, the DML statement specified in `dml-derived-table` is executed first, and the rows affected by that DML are materialized into a temporary table whose columns are described by the REFERENCING clause. The temporary table represents the result set of `dml-derived-table`.

Use REFERENCING ( ) or REFERENCING (NONE) if the results do not need to be materialized into a temporary table because you are not referencing them in the query.

If you specify REFERENCING ( ) or REFERENCING (NONE), then the updated rows are not materialized into a temporary table that represents the result set of `dml-derived-table` because they are not being

referenced in the query. The temporary table in this case is an empty table. Use this feature if you want `dml-statement` to be executed before the main statement is executed.

In the results, OLD columns contain the values as seen by the scan that finds the rows to include in the update operation. FINAL columns contain the values after referential integrity checks have been made, computed and default columns have been updated, and all triggers have been fired (excluding AFTER triggers of type FOR STATEMENT).

Statement	Supported table versions
INSERT	FINAL
DELETE	OLD
UPDATE	FINAL and/or OLD
MERGE	FINAL and/or OLD

When specifying both OLD and FINAL names, two correlation names are used; however, these are not true correlations since they both refer to the same result set. If you specify `REFERENCING (OLD AS O FINAL AS F )`, then there is an implicit join predicate: `O.rowid = F.rowid`.

The INSERT statement only supports FINAL. Consequently, the values of updated rows that are modified by an INSERT ON EXISTING UPDATE statement do not appear in the result set of the derived table. Instead, use the MERGE statement to perform the insert-else-update processing.

The `dml-derived-table` statement can only reference one updatable table; updates over multiple tables return an error. Also, selecting from `dml-statement` is not allowed if the DML statement appears inside a correlated subquery or common table expression because the semantics of these constructs can be unclear.

openxml-operator

Use this parameter to return a result set from an XML document, by using the OPENXML operator.

array-operator

Use this parameter to return a result set from an ARRAY, by using an array operator such as UNNEST.

WITH table-hint clause

The WITH `table-hint` clause allows you to specify the behavior to be used only for this table, and only for this statement. Use this clause to change the behavior without changing the isolation level or setting a database or connection option. Table hints can be used for base tables, temporary tables, and materialized views.

> ⚠ Caution
>
> The WITH `table-hint` clause is an advanced feature that should be used only if needed, and only by experienced database administrators. In addition, the setting may not be respected in all situations.

Isolation level related table hints

Isolation level table hints are used to specify isolation level behavior when querying tables. They specify a locking method that is used only for the specified tables, and only for the current query. You cannot specify snapshot isolation levels as table hints.

Table hint	Description
HOLDLOCK	Sets the behavior to be equivalent to isolation level 3. This table hint is synonymous with SERIALIZABLE.
NOLOCK	Sets the behavior to be equivalent to isolation level 0. This table hint is synonymous with READUNCOMMITTED.
READCOMMITTED	Sets the behavior to be equivalent to isolation level 1.
READPAST	Instructs the database server to ignore, instead of block on, write-locked rows. This table hint can only be used with isolation level 1. The READPAST hint is respected only when the correlation name in the FROM clause refers to a base or globally shared temporary table. In other situations (views, proxy tables, and table functions) the READPAST hint is ignored. Queries within views may utilize READPAST as long as the hint is specified for a correlation name that is a base table. Using the READPAST table hint can lead to anomalies due to the interaction of locking and predicate evaluation within the server. In addition, you cannot use the READPAST hint against tables that are the targets of a DELETE, INSERT, or UPDATE statement.
READUNCOMMITTED	Sets the behavior to be equivalent to isolation level 0. This table hint is synonymous with NOLOCK.
REPEATABLEREAD	Sets the behavior to be equivalent to isolation level 2.
SERIALIZABLE	Sets the behavior to be equivalent to isolation level 3. This table hint is synonymous with HOLDLOCK.
UPDLOCK	Indicates that rows processed by the statement from the hinted table are locked using intent locks. The affected rows remain locked until the end of the transaction. UPDLOCK works at all isolation levels and uses intent locks.
XLOCK	Indicates that rows processed by the statement from the hinted table are to be locked exclusively. The affected rows remain locked until the end of the transaction. XLOCK works at all isolation levels and uses write locks.

i Note

If you are writing queries for databases that participate in MobiLink synchronization, it is recommended that you do not use the READPAST table hint in your synchronization scripts.

If you are considering READPAST because your application performs many updates that affect download performance, an alternative solution is to use snapshot isolation.

Optimization table hint (FASTFIRSTROW)

The FASTFIRSTROW table hint allows you to set the optimization goal for the query without setting the optimization_goal option to First-row. When you use FASTFIRSTROW, the database server chooses an access plan that is intended to reduce the time to fetch the first row of the query's result.

WITH (index-hint) clause

The WITH (`index-hint`) clause allows you to specify index hints that override the query optimizer plan selection algorithms, and tell the optimizer exactly how to access the table using indexes. Index hints can be used for base tables, temporary tables, and materialized views.

NO INDEX

Use this clause to force a sequential scan of the table (indexes are not used). Sequential scans may be very costly.

INDEX ([PRIMARY KEY | FOREIGN KEY] index-name [,...])

Use this clause to specify up to four indexes that the optimizer must use to satisfy the query.

If any of the specified indexes cannot be used, an error is returned.

You can specify PRIMARY KEY or FOREIGN KEY to remove ambiguity in the cases where the PRIMARY KEY index and FOREIGN KEY index on a table have the same name.

If you specify an index name in the index hint without the PRIMARY or FOREIGN key, and multiple indexes with the same name exist on a table, the optimizer chooses the normal index. If a normal index does not exist, the optimizer chooses the primary key index. If a primary key index does not exist, the foreign key index is used instead.

`index-name` can be qualified by specifying the user ID and the table name of the index.

The indexes specified in the INDEX clause must be indexes defined for that table; otherwise, an error is returned. For example, `FROM Products WITH( INDEX (Products.xx))` returns an error if the index xx is not defined for the Products table. Likewise, `FROM Products WITH( INDEX (sales_order_items.sales_order_items))` returns an error because the sales_order_items.sales_order_items index exists but is not defined for the Products table.

INDEX ONLY { ON | OFF }

Use this clause to control whether an index-only retrieval of data is performed. If the INDEX (`index-name`...) clause is specified with INDEX ONLY ON, the database server attempts an index-only retrieval using the specified indexes. If any of the specified indexes cannot be used in satisfying an index-only retrieval, an error is returned (for example, if there are no indexes, or if the existing indexes cannot satisfy the query).

Specify INDEX ONLY OFF to prevent an index-only retrieval.

FORCE INDEX (index-name)

The FORCE INDEX (`index-name`) syntax is provided for compatibility, and does not support specifying more than one index. This clause is equivalent to `WITH ( INDEX ( index-name))`.

CLUSTERED INDEX

Use this clause to specify that the optimizer must use a clustered index if one exists. The index name is not specified as only one clustered index can exist for a base table. If a clustered index doesn't exist or it cannot be used, an error is returned.

Remarks

Subqueries and subselects are supported as arguments to stored procedures and table functions in the FROM clause. For example, the following FROM clause is valid:

```
SELECT *, ( SELECT 12 x ) D
FROM sa_rowgenerator( 1,( SELECT 12 x ) );
```

The SELECT, UPDATE, and DELETE statements require a table list to specify which tables are used by the statement.

> **i** Note
>
> Although the FROM clause description refers to tables, it also applies to views and derived tables unless otherwise noted.

The FROM clause creates a result set consisting of all the columns from all the tables specified. Initially, all combinations of rows in the component tables are in the result set, and the number of combinations is usually reduced by JOIN conditions and/or WHERE conditions.

You cannot use an ON phrase with CROSS JOIN.

Privileges

The FILE clause of `openstring-expression` requires the READ FILE privilege.

The TABLE clause of `openstring-expression` requires the user to own the referenced tables, or to have the SELECT ANY TABLE privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The FROM clause is a fundamental part of the ANSI/ISO SQL Standard. The complexity of the FROM clause means that you should check individual components of a FROM clause against the appropriate portions of the standard. The following is a non-exhaustive list of optional ANSI/ISO SQL Language Features supported in the software:

- CROSS JOIN, FULL OUTER JOIN, and NATURAL JOIN constitute optional ANSI/ISO SQL Feature F401.
- INTERSECT and INTERSECT ALL constitute optional ANSI/ISO SQL Feature F302.

- EXCEPT ALL is optional ANSI/ISO SQL Language Feature F304.
- derived tables are ANSI/ISO SQL Language Feature F591.
- procedures in the FROM clause (table functions) are ANSI/ISO SQL Feature T326. The ANSI/ISO SQL Standard requires the keyword TABLE to identify the output of a procedure as a table expression, whereas in the software, the TABLE keyword is unnecessary.
- common table expressions are optional ANSI/ISO SQL Language Feature T121. Using a common table expression in a derived table nested within another common table expression is Language Feature T122.
- recursive table expressions are ANSI/ISO SQL feature T131. Using a recursive table expression in a derived table nested within a common table expression is optional ANSI/ISO SQL Language Feature T132.

The following components of the FROM clause are not in the standard:

- KEY JOIN.
- CROSS APPLY and OUTER APPLY.
- OPENSTRING.
- a `table-expression` using CONTAINS (full text search).
- specifying a `dml-statement` as a derived table.
- all table hints, including the use of WITH, FORCE INDEX, READPAST and isolation level hints.
- LATERAL (`table-expression`). LATERAL (`select-statement`) is in the ANSI/ISO SQL Standard as optional ANSI/ISO SQL Language Feature T491.

Example

The following are valid FROM clauses:

```
...
FROM GROUPO.Employees
...
```

```
...
FROM GROUPO.Employees NATURAL JOIN GROUPO.Departments
...
```

```
...
FROM GROUPO.Customers
KEY JOIN GROUPO.SalesOrders
KEY JOIN GROUPO.SalesOrderItems
KEY JOIN GROUPO.Products
...
```

```
...
FROM GROUPO.Employees CONTAINS ( Street, ' Way ' )
...
```

The following query illustrates how to use derived tables in a query:

```
SELECT Surname, GivenName, number_of_orders
FROM GROUPO.Customers JOIN
     ( SELECT CustomerID, COUNT(*)
```

```
       FROM GROUPO.SalesOrders
        GROUP BY CustomerID )
     AS sales_order_counts( CustomerID,
                            number_of_orders )
ON ( Customers.ID = sales_order_counts.CustomerID )
WHERE number_of_orders > 3;
```

The following query illustrates how to select rows from stored procedure result sets:

```
SELECT t.ID, t.QuantityOrdered AS q, p.name
FROM GROUPO.ShowCustomerProducts( 149 ) t JOIN GROUPO.Products p
ON t.ID = p.ID;
```

The following example illustrates how to perform a query by using the OPENSTRING clause to query a file. The CREATE TABLE statement creates a table called testtable with two columns, column1 and columns2. The UNLOAD statement creates a file called `testfile.dat` by unloading rows from the RowGenerator table. The SELECT statement uses the OPENSTRING clause in a FROM clause to query `testfile.dat` using the schema information from both the testtable and RowGenerator tables. The query returns one row with the value 49.

```
CREATE TABLE testtable( column1 CHAR(10), column2 INT );
UNLOAD SELECT * FROM RowGenerator TO 'testfile.dat';
SELECT A.column2
  FROM OPENSTRING( FILE 'testfile.dat' )
  WITH ( TABLE testtable( column2 ) ) A, RowGenerator B
  WHERE A.column2 = B.row_num
  AND A.column2 < 50
  AND B.row_num > 48;
```

The following example illustrates how to perform a query using the OPENSTRING clause to query a string value. The SELECT statement uses the OPENSTRING clause in a FROM clause to query a string value using the schema information provided in the WITH clause. The query returns two columns with three rows.

```
SELECT *
  FROM OPENSTRING( VALUE '1,"First"$2,"Second"$3,"Third"')
  WITH (c1 INT, c2 VARCHAR(30))
  OPTION ( DELIMITED BY ',' ROW DELIMITED BY '$')
  AS VALS
```

The following example illustrates how to perform a query to select the rows modified by a data manipulation statement. In this example, a warning is issued when the stock of blue items drops by more than half.

```
SELECT old_products.name, old_products.quantity, final_products.quantity
FROM
( UPDATE GROUPO.Products SET quantity = quantity - 10 WHERE color = 'Blue' )
REFERENCING ( OLD AS old_products FINAL AS final_products )
WHERE final_products.quantity < 0.5 * old_products.quantity;
```

The following query illustrates the use of the WITH clause when selecting from a fictitious procedure called ShowCustomerProducts:

```
SELECT sp.ident, sp.quantity, Products.name
FROM GROUPO.ShowCustomerProducts( 149 )
   WITH ( ident INT, description CHAR(20), quantity INT ) sp
JOIN GROUPO.Products
ON sp.ident = Products.ID;
```

Related Information

Text Index Concepts and Reference
Row Locks
What to Specify When Creating or Altering Text Configuration Objects
Example Text Configuration Objects
SELECT Over a DML Statement
Data Manipulation Statements
Isolation Levels and Consistency
MobiLink Isolation Levels
The FROM Clause: Specifying Tables
Joins: Retrieving Data from Several Tables
%TYPE and %ROWTYPE Attributes [page 115]
PIVOT Clause [page 1300]
UNPIVOT Clause [page 1459]
Named Parameters [page 128]
download_cursor Table Event
download_delete_cursor Table Event
upload_fetch Table Event
CONTAINS Search Condition [page 76]
optimization_goal Option
UNNEST Array Operator [page 28]
OPENXML Operator [page 20]
LOAD TABLE Statement [page 1247]
DELETE Statement [page 1070]
SELECT Statement [page 1362]
UPDATE Statement [page 1463]
INSERT Statement [page 1232]
MERGE Statement [page 1271]

1.4.4.173 GET DATA Statement [ESQL]

Gets string or binary data for one column of the current row of a cursor.

Syntax

```
GET DATA cursor-name
COLUMN column-num
OFFSET start-offset
[ WITH TEXTPTR ]
{ USING DESCRIPTOR sqlda-name | INTO hostvar, ... }
```

```
cursor-name : identifier | hostvar
```

```
column-num : integer | hostvar
```

```
start-offset : integer | hostvar

sqlda-name : identifier
```

Parameters

COLUMN clause

The value of `column-num` starts at one, and identifies the column whose data is to be fetched. That column must be of a string or binary type.

OFFSET clause

The `start-offset` indicates the number of bytes to skip over in the field value. Normally, this would be the number of bytes previously fetched. The number of bytes fetched on this GET DATA statement is determined by the length of the target host variable.

WITH TEXTPTR clause

If the WITH TEXTPTR clause is given, a text pointer is retrieved into a second host variable or into the second field in the SQLDA. This text pointer can be used with the Transact-SQL READ TEXT and WRITE TEXT statements. The text pointer is a 16-bit binary value, and can be declared as follows:

```
DECL_BINARY( 16 ) textptr_var;
```

The WITH TEXTPTR clause can only be used with long data types (LONG BINARY, LONG VARCHAR, TEXT, IMAGE). If you attempt to use it with another data type, the error INVALID_TEXTPTR_VALUE is returned.

The total length of the data is returned in the SQLCOUNT field of the SQLCA structure.

USING DESCRIPTOR clause

The `sqlda-name` specifies the SQLDA (SQL Descriptor Area) that receives the fetched data. The USING DESCRIPTOR clause provides a dynamic method of specifying host variables to receive fetched data.

INTO clause

Use the INTO clause to specify the host variable that receives the fetched data. The indicator value for the target host variable is of type a_sql_len, which is currently a 16-bit value, so it is not always large enough to contain the number of bytes truncated. Instead, it contains a negative value if the field contains the NULL value, a positive value (not necessarily the number of bytes truncated) if the value is truncated, and zero if a non-NULL value is not truncated.

Similarly, if a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY host variable is used with an offset greater than zero, the untrunc_len field does not accurately indicate the size before truncation.

Remarks

Get a piece of one column value from the row at the current cursor position. The cursor must be opened and positioned on a row, using FETCH.

GET DATA is usually used to fetch LONG BINARY or LONG VARCHAR fields.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example uses GET DATA to fetch a binary large object (also called a BLOB).

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY(1000) piece;
short ind;
EXEC SQL END DECLARE SECTION;
int size;
/* Open a cursor on a long varchar field */
EXEC SQL DECLARE big_cursor CURSOR FOR
SELECT long_data FROM some_table
WHERE key_id = 2;
EXEC SQL OPEN big_cursor;
EXEC SQL FETCH big_cursor INTO :piece;
for( offset = 0; ; offset += piece.len ) {
   EXEC SQL GET DATA big_cursor COLUMN 1
   OFFSET :offset INTO :piece:ind;
   /* Done if the NULL value */
   if( ind < 0 ) break;
   write_out_piece( piece );
   /* Done when the piece was not truncated */
   if( ind == 0 ) break;
}
EXEC SQL CLOSE big_cursor;
```

Related Information

FETCH Statement [ESQL] [SP] [page 1162]
READTEXT Statement [T-SQL] [page 1320]
SET Statement [page 1373]

1.4.4.174 GET DESCRIPTOR Statement [ESQL]

Retrieves information about a variable within a descriptor area, or retrieves its value.

Syntax

```
GET DESCRIPTOR descriptor-name
{ hostvar = COUNT | VALUE { integer | hostvar } assignment, ... }
```

```
assignment :
 hostvar =
TYPE
| LENGTH
| PRECISION
| SCALE
| DATA
| INDICATOR
| NAME
| NULLABLE
| RETURNED_LENGTH
```

```
descriptor-name : identifier
```

Remarks

The GET DESCRIPTOR statement is used to retrieve information about a variable within a descriptor area, or to retrieve its value.

The value { `integer` | `hostvar` } specifies the variable in the descriptor area about which the information is retrieved. Type checking is performed when doing GET...DATA to ensure that the host variable and the descriptor variable have the same data type. LONG VARCHAR and LONG BINARY are not supported by GET DESCRIPTOR...DATA.

If an error occurs, it is returned in the SQLCA.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

GET DESCRIPTOR is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL".

Example

The following example returns the type of the column with position col_num in sqlda.

```
int get_type( SQLDA *sqlda, int col_num )
{
    EXEC SQL BEGIN DECLARE SECTION;
    int ret_type;
    int col = col_num;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL GET DESCRIPTOR sqlda VALUE :col :ret_type = TYPE;
    return( ret_type );
}
```

Related Information

The SQL Descriptor Area (SQLDA)
ALLOCATE DESCRIPTOR Statement [ESQL] [page 660]
DEALLOCATE DESCRIPTOR Statement [ESQL] [page 1054]
SET DESCRIPTOR Statement [ESQL] [page 1380]

1.4.4.175 GET OPTION Statement [ESQL]

Gets the current setting of an option. It is recommended that you use the CONNECTION_PROPERTY function instead.

Syntax

```
GET OPTION [ userid.]option-name
{ INTO hostvar | USING DESCRIPTOR sqlda-name }
```

```
userid : identifier, string | hostvar
```

```
option-name :
identifier
| string
| hostvar
```

```
hostvar : indicator variable allowed
```

```
sqlda-name : identifier
```

Remarks

The GET OPTION statement is provided for compatibility with older versions of the software. The recommended way to get the values of options is to use the CONNECTION_PROPERTY system function.

The GET OPTION statement gets the option setting of the option `option-name` for the user `userid` or for the connected user if `userid` is not specified. This is either the user's personal setting or the PUBLIC setting if there is no setting for the connected user. If the option specified is a database option and the user has a temporary setting for that option, then the temporary setting is retrieved.

If `option-name` does not exist, GET OPTION returns the warning SQLE_NOTFOUND.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement illustrates use of GET OPTION.

```
EXEC SQL GET OPTION 'date_format' INTO :datefmt;
```

Related Information

Alphabetical List of System Procedures [page 1520]
SET OPTION Statement [page 1387]
CONNECTION_PROPERTY Function [System] [page 293]

1.4.4.176 GOTO Statement

Branches to a labeled statement.

Syntax

```
label-name:
sql-statement(s)
GOTO label-name
```

Remarks

Statements in a procedure, trigger, or batch can be labeled using a valid identifier followed by a colon (for example mylabel:), provided that the label is at the beginning of a loop, conditional, or block. The label can then be referenced in a GOTO statement, causing the execution point to move to the top of the loop/condition or the first statement within the block.

When referencing a label in a GOTO statement, do not specify the colon.

If you nest compound statements, then you can only go to labels within the current compound statement and any of its ancestor compound statements. You cannot go to labels located in other compound statements that are nested within the ancestors.

The database server supports the use of the GOTO statement in Transact-SQL procedures, triggers, or batches. In Transact-SQL, label use is not restricted to the beginning of loops, conditionals, or blocks; they can occur on any statement. However, the same restrictions apply to using the GOTO statement within nested compound statements.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In the following example, if the GotoTest procedure is executed, then the GOTO lbl1 repositions execution to the SET i2 = 200 statement. The returned values for column i2 in the result are 203 for all 5 rows in the result set.

```
CREATE OR REPLACE PROCEDURE GotoTest()
RESULT ( id INT, i INT, i2 INT )
BEGIN
    DECLARE LOCAL TEMPORARY TABLE gotoTable( id INT DEFAULT AUTOINCREMENT, i
INT, i2 INT ) NOT TRANSACTIONAL;
    DECLARE i INT;
    DECLARE i2 INT;
    SET i = 100;
    lbl1: WHILE i < 105 LOOP
        SET i2 = 200;
        SET i2 = i2 + 1;
        lbl2: BEGIN
            SET i2 = i2 + 1;
            lbl3: BEGIN
                SET i2 = i2 + 1;
                INSERT INTO gotoTable(i, i2) VALUES(i, i2);
                SET i = i + 1;
                IF( i < 110 ) THEN
                    GOTO lbl1
                END IF;
            END
        END
    END LOOP;
    SELECT id, i, i2 FROM gotoTable ORDER BY id;
END;
CALL GotoTest();
```

Table 4: Results

id	i	i2
1	100	203
2	101	203
3	102	203
4	103	203
5	104	203

If the GotoTest procedure is changed to use GOTO lbl2 instead of GOTO lbl1, then the GOTO statement repositions execution to the SET i2 = i2 + 1 statement immediately after the lbl2: BEGIN statement, and the returned values in column i2 become 203, 205, 207, up to 221.

If the GotoTest procedure is changed to use GOTO lbl3, then the GOTO statement repositions execution to the SET i2 = i2 +1 statement immediately after the lbl3: BEGIN statement, and the returned values in column i2 become 203, 204, 205, up to 212.

The following Transact-SQL batch prints the message "yes" in the database server messages window four times:

```
DECLARE @count SMALLINT
SELECT @count = 1
restart:
   PRINT 'yes'
   SELECT @count = @count + 1
   WHILE @count <=4
    GOTO restart;
```

1.4.4.177 GRANT Statement

Grant system and object-level privileges to users and roles.

Syntax

Grant system privileges

```
GRANT system-privilege [,...]
TO grantee [ ,... ]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

Grant object-level privileges

```
GRANT object-level-privilege,...
 ON [ owner.]object-name
 TO to-userid,...
[ WITH GRANT OPTION ]
```

```
object-level-privilege :
ALL [ PRIVILEGES ]
| ALTER
| DELETE
| INSERT
| LOAD
| REFERENCES [ ( column-name,... ) ]
| SELECT [ ( column-name,... ) ]
| TRUNCATE
| UPDATE [ ( column-name,... ) ]
```

Grant the SET USER system privilege

```
GRANT SET USER [ ( user-list | ANY [ WITH ROLES role-list ] ) ]
TO grantee [,...]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

Grant the CHANGE PASSWORD system privilege

```
GRANT CHANGE PASSWORD [ ( user-list | ANY [ WITH ROLES role-list ] ) ]
TO grantee [,...]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

Parameters

grantee

The user ID of a user, or the name of a role. You cannot grant privileges to compatibility roles. You can grant privileges to any system role; however, only the following system roles support logins: PUBLIC, dbo, diagnostics, rs_systabgroup, and SA_DEBUG

object-level-privilege

ALL privilege

This privilege grants ALTER, DELETE, INSERT, REFERENCES, SELECT, and UPDATE privileges on tables. This privilege grants DELETE, INSERT, and UPDATE privileges on views.

ALTER privilege

This privilege allows the user to alter the named table with the ALTER TABLE statement. This privilege is not allowed for views.

DELETE privilege

This privilege allows the user to delete rows from the named table or view.

INSERT privilege

This privilege allows the user to insert rows into the named table or view.

LOAD privilege

This privilege allows the user to load the named table or view.

REFERENCES privilege

This privilege allows the user to create indexes on the named table and on the foreign keys that reference the named tables. If column names are specified, the user can reference only those columns. REFERENCES privileges on columns cannot be granted for views, only for tables. INDEX is a synonym for REFERENCES.

SELECT privilege

This privilege allows the user to view information in the view or table. If column names are specified, the users are allowed to view only those columns. SELECT privileges on columns cannot be granted for views, only for tables.

TRUNCATE privilege

This privilege allows the user to truncate the named object.

UPDATE privilege

This privilege allows the user to update rows in the view or table. If column names are specified, the user can update only those columns.

WITH GRANT OPTION

If WITH GRANT OPTION is specified, then the named user ID is also given permission to GRANT the same privilege to other user IDs. Users who can exercise a role do not inherit the WITH GRANT OPTION if it is granted to a role.

GRANT SET USER

user-list

Specify the comma-separated list of users that `grantee-list` can impersonate. For example: `GRANT SET USER(u1, u2, u3)...`

ANY [WITH ROLES role_list] clause

Specify who `grantee` can impersonate without providing specific user IDs.

If just ANY is specified, then the user can impersonate any other user. This is the default.

If ANY WITH ROLES `role-list` is specified, users in `grantee-list` can impersonate anyone who has at least one of the user-defined roles listed in `role-list`, where `role-list` is a comma-separated list of user-defined roles.

WITH ADMIN [ONLY] OPTION clause

The WITH ADMIN OPTION and WITH ADMIN ONLY OPTION clauses can only be specified with the ANY clause.

GRANT CHANGE PASSWORD

user-list

Specify a comma-separated list of users that `grantee` can change passwords for.

ANY [WITH ROLES role-list] clause

Specify who `grantee` can change the password for without providing specific user IDs.

If just ANY is specified, then the user can change any user's password. This is the default.

If ANY WITH ROLES `role-list` is specified, the `grantee` can change the password for anyone who has at least one of the user-defined roles listed in `role-list`, where `role-list` is a comma-separated list of user-defined roles.

WITH ADMIN [ONLY] OPTION clause

The WITH ADMIN OPTION and WITH ADMIN ONLY OPTION clauses can only be specified with the ANY clause.

Remarks

You can grant privileges on disabled objects. Privileges on disabled objects are stored in the database and become effective when the object is enabled.

With the exception of the SYS role, you can grant/revoke additional privileges to/from a system role, provided you have administrative rights on the privileges you are granting/revoking.

Granting SET USER to a user multiple times, specifying different user IDs they can impersonate, grants additional users to the list they can impersonate (as opposed to overwriting the previous grants).

Granting impersonation rights (GRANT SET USER) is not an indication of whether a user can successfully impersonate another user. Evaluation of whether a user can impersonate another user is done when the user ID attempts to start impersonating another user by executing a SETUSER statement. The impersonating user must have the SET USER system privilege, and must meet the at-least criteria required for impersonation.

The GRANT syntax related to authorities, permissions, and groups used in pre-16.0 versions of the software is still supported but deprecated.

When granting system privileges to the MANAGE ROLES system privilege, you must use the special internal representation for MANAGE ROLES, which is SYS_MANAGE_ROLES_ROLE (for example, `GRANT privilege-name TO SYS_MANAGE_ROLES_ROLE`)

Privileges

You must have administrative rights for each privilege you grant.

To grant object-level privileges, you must also have the MANAGE ANY OBJECT PRIVILEGE system privilege, with administrative rights.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Granting system privileges to roles

To grant the CREATE ANY OBJECT system privilege to the role RoleA without giving RoleA administrative rights, execute the following statement:

```
GRANT CREATE ANY OBJECT TO RoleA;
```

To grant RoleA the CREATE ANY OBJECT system privilege along with the ability to grant or revoke the system privilege to and from other users and roles, execute the following statement:

```
GRANT CREATE ANY OBJECT TO RoleA WITH ADMIN OPTION;
```

To give RoleA administrative rights to the BACKUP DATABASE system privilege, but not the ability to use the BACKUP DATABASE privilege, execute the following statement:

```
GRANT BACKUP DATABASE TO RoleA WITH ADMIN ONLY OPTION;
```

Granting SET USER to users

The following example specifies that User4 and User5 can impersonate User1, User2, and User3.

```
GRANT SET USER ( User1, User2, User3 ) TO User4, User5;
```

The following example specifies that User1 can impersonate any user in the database. As well, User1 can grant the SET USER system privilege to other users.

```
GRANT SET USER (ANY) TO User1 WITH ADMIN OPTION;
```

The following example specifies that User1 can impersonate any user who has been granted the READ_ROLE or MODIFY_ROLE role.

```
GRANT SET USER (ANY WITH ROLES READ_ROLE, MODIFY_ROLE) TO User1;
```

Related Information

Replication-related System Roles
Initiation of Synchronization
System Privileges
Roles
Compatibility Roles
User-defined Roles
Impersonation
Changes in Inheritance Behavior for Some Authorities That Became Roles
SETUSER Statement [page 1403]
GRANT Statement (Authorities and Groups) (Deprecated)
REVOKE Statement [page 1345]

1.4.4.178 GRANT CONNECT Statement

Creates a new user, and can also be used by a user to change their own password. However, it is recommended that you use the CREATE USER statement to create users instead of the GRANT CONNECT statement.

Syntax

```
GRANT CONNECT TO userid, ...
[ IDENTIFIED BY password, ... ]
```

Parameters

userid

Specify the name of the user you are creating. The user name is an identifier which must follow the rules for an identifier. More than one user name can be specified using commas.

password

Specify the password of the user. You can specify an identifier or a string literal as the password. If you specify an identifier then you must follow the rules for an identifier. If you specify a string literal, then there are fewer restrictions. Specify one password for each user using commas.

Remarks

You do not have to specify an IDENTIFIED BY clause for the user. A user with no password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID. To create a user with no password, use:

```
GRANT CONNECT TO userid;
```

The verify_password_function option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement.

Subject to the restrictions of minimum password length, a user can be created with an empty password, in which case they do not need to specify a password when logging in. This is not the same as creating a user with no password. Creating such a user is not advisable.

```
GRANT CONNECT TO userid IDENTIFIED BY "";
```

If you use the GRANT CONNECT statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must either be changing your own password using GRANT CONNECT, or have the MANAGE ANY USER privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following examples create a user named SQLTester, each with the same password Welcome. All of these examples specify a valid identifier for the password.

```
GRANT CONNECT TO SQLTester IDENTIFIED BY Welcome;
GRANT CONNECT TO SQLTester IDENTIFIED BY "Welcome";
GRANT CONNECT TO SQLTester IDENTIFIED BY `Welcome`;
GRANT CONNECT TO SQLTester IDENTIFIED BY [Welcome];
```

The following example creates a user named SQLTester with the password Wel[come]. This example specifies a string literal for the password which permits the use of square brackets.

```
GRANT CONNECT TO SQLTester IDENTIFIED BY 'Wel[come]';
```

The following example creates two users named SQLTester1 and SQLTester2 with the passwords Wel[come]1 and Wel[come]2 respectively. This example specifies string literals for the passwords which permits the use of square brackets.

```
GRANT CONNECT TO SQLTester1, SQLTester2 IDENTIFIED BY 'Wel[come]1', 'Wel[come]2';
```

Related Information

Password and User ID Restrictions and Considerations
CREATE USER Statement [page 1045]
verify_password_function Option

1.4.4.179 GRANT CONSOLIDATE Statement [SQL Remote]

Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

Syntax

```
GRANT CONSOLIDATE
TO userid
TYPE message-system, ...
ADDRESS address-string, ...
[ SEND { EVERY | AT } hh:mm:ss ]
```

```
message-system :
FILE | FTP | SMTP
```

```
address : string
```

Parameters

userid

The user ID for the user to be granted the privileges.

message-system

One of the message systems supported by SQL Remote.

address

The address for the specified message system.

Remarks

In a SQL Remote installation, the database immediately above the current database in a SQL Remote hierarchy must be granted CONSOLIDATE privilege. GRANT CONSOLIDATE is issued at a remote database to identify its consolidated database. Each database can have only one user ID with CONSOLIDATE privileges: you cannot have a database that is a remote database for more than one consolidated database.

The consolidated user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes. There can be only one consolidated user per remote database.

For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

The GRANT CONSOLIDATE statement is required for the consolidated database to receive messages, but does not by itself subscribe the consolidated database to any data. To subscribe to data, a subscription must be created for the consolidated user ID to one of the publications in the current database. Running the database extraction utility at a consolidated database creates a remote database with the proper GRANT CONSOLIDATE statement already executed.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted remote privileges without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. To run the Message Agent continuously, you must ensure that every user with REMOTE privilege has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many remote databases, the Message Agent is run periodically, and that the consolidated database will have no SEND clause specified.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement grants consolidated status to the Sam_Singer user ID:

```
GRANT CONSOLIDATE TO Sam_Singer
TYPE SMTP
ADDRESS 'Singer, Samuel';
```

Related Information

SQL Remote Message Systems
Granting CONSOLIDATE Privilege (SQL Central)
GRANT PUBLISH Statement [SQL Remote] [page 1206]
GRANT REMOTE Statement [SQL Remote] [page 1208]
REVOKE CONSOLIDATE Statement [SQL Remote] [page 1348]
ALTER REMOTE [MESSAGE] Statement [SQL Remote] [page 708]
CREATE REMOTE [MESSAGE] Statement [SQL Remote] [page 950]
CREATE SUBSCRIPTION Statement [SQL Remote] [page 993]

1.4.4.180 GRANT CREATE Statement

Allows a user to create database objects in the specified dbspace.

Syntax

```
GRANT CREATE ON dbspace-name
TO userid, ...
```

Remarks

When you initialize a database, it contains one database file. This first database file is called the main file. All database objects and all data are placed, by default, in the main file. A dbspace is an additional database file that creates more space for data. A database can be held in up to 13 separate files (the main file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

Privileges

You must have the MANAGE ANY USER privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Database File Types

1.4.4.181 GRANT EXECUTE Statement

Grants a user privilege to execute a procedure or user-defined function.

Syntax

```
GRANT EXECUTE ON [ owner.]{ procedure-name | user-defined-function }
TO userid, ...
```

Remarks

None.

Privileges

You must own the procedure, or have the MANAGE ANY OBJECT PRIVILEGE system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Core Feature.

Example

The following example allows user SQLTester to execute the Calculate_Report procedure.

```
GRANT EXECUTE ON Calculate_Report
TO SQLTester;
```

The following SQL statement grants M_Haneef the privilege required to execute a procedure named my_procedure.

```
GRANT EXECUTE
ON my_procedure
TO M_Haneef;
```

Related Information

System Procedures [page 1511]

1.4.4.182 GRANT INTEGRATED LOGIN Statement

Grants a user privilege to execute a procedure or user-defined function.

Syntax

```
GRANT INTEGRATED LOGIN TO user-profile-name, ...
[ AS USER userid ]
```

Remarks

The GRANT INTEGRATED LOGIN statement creates an explicit integrated login mapping between one or more Windows user or group profiles and an existing database user ID, allowing users who successfully log in to their local computer to connect to a database without having to provide a user ID or password. The `user-profile-name` can be of the form `domain\user-name`. The database user ID the integrated login is mapped to must have a password.

Privileges

You must have the MANAGE ANY USER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Microsoft Windows Integrated Login

1.4.4.183 GRANT KERBEROS LOGIN Statement

Creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID.

Syntax

```
GRANT KERBEROS LOGIN TO client-Kerberos-principal, ...
AS USER userid
```

Remarks

The GRANT KERBEROS LOGIN statement creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID. This login mapping allows users who have successfully logged in to Kerberos (users who have a valid Kerberos ticket-granting ticket) to connect to a database without having to provide a user ID or password.

To use the GRANT KERBEROS LOGIN statement:

- You must already have Kerberos configured to use SQL Anywhere.
- You must already have your SQL Anywhere database configured to use Kerberos.
- The database user and the Kerberos principal must already exist.
- The database user ID the Kerberos login is mapped to must have a password.
- The `client-Kerberos-principal` must have the format `user/instance@REALM`, where `/instance` is optional. The full principal, including the realm, must be specified, and principals that differ only in the instance or realm are treated as different.
- Principals are case sensitive and must be specified in the correct case. Mappings for multiple principals that differ only in case are not supported (for example, you cannot have mappings for both jjordan@MYREALM.COM and JJordan@MYREALM.COM).
- If no explicit mapping is made for a Kerberos principal, and the Guest database user ID exists and has a password, then the Kerberos principal connects using the Guest database user ID (the same Guest database user ID as for integrated logins).

Privileges

You must have the MANAGE ANY USER privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following SQL statement grants database login privilege to a fictitious Kerberos user pchin.

```
GRANT KERBEROS LOGIN TO "pchin@MYREALM.COM"
AS USER "kerberos-user";
```

Related Information

Kerberos User Authentication

1.4.4.184 GRANT PUBLISH Statement [SQL Remote]

Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.

Syntax

GRANT PUBLISH TO `userid`

Remarks

Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the publisher. The GRANT PUBLISH statement sets the publisher user ID associated with these outgoing messages. To change publishers, you must revoke publisher rights from the current publisher (REVOKE PUBLISH), and then grant them to the new publisher.

The database publisher can be determined by querying the special value CURRENT PUBLISHER as follows:

```
SELECT CURRENT PUBLISHER;
```

If there is no publisher, the value of CURRENT PUBLISHER is NULL.

The CURRENT PUBLISHER special value can be used as a default setting for columns. It is often useful to have a CURRENT PUBLISHER column as part of the primary key for replicating tables, as this configuration helps prevent primary key conflicts due to updates at more than one site.

To change the publisher, you must first drop the current publisher using the REVOKE PUBLISH statement, and then create a new publisher using the GRANT PUBLISH statement.

Executing this statement changes the value of the PUBLIC.db_publisher database option.

Privileges

You must have the SET ANY SYSTEM OPTION system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement sets the database publisher to user ID JohnS.

```
GRANT PUBLISH TO JohnS;
```

The following statements revoke database publisher from JohnS and grant it to IrisM.

```
REVOKE PUBLISH FROM JohnS;
GRANT PUBLISH TO IrisM;
```

Related Information

Creating a Publisher (SQL Central)
GRANT ROLE Statement [page 1210]
db_publisher Option

GRANT CONSOLIDATE Statement [SQL Remote] [page 1199]
REVOKE PUBLISH Statement [SQL Remote] [page 1349]
CREATE SUBSCRIPTION Statement [SQL Remote] [page 993]
CURRENT PUBLISHER Special Value [page 91]
CREATE REMOTE [MESSAGE] Statement [SQL Remote] [page 950]
GRANT REMOTE Statement [SQL Remote] [page 1208]

1.4.4.185 GRANT REMOTE Statement [SQL Remote]

Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.

Syntax

```
GRANT REMOTE TO userid, ...
TYPE message-system, ...
ADDRESS address-string, ...
[ SEND { EVERY | AT } send-time ]
```

Parameters

userid

The user ID for the user to be granted the privilege

message-system

One of the message systems supported by SQL Remote. It must be one of the following values:

- FILE
- FTP
- SMTP

address-string

A string containing a valid address for the specified message system.

send-time

A string containing a time specification in the form `hh:mm:ss`.

Remarks

In a SQL Remote installation, each database receiving messages from the current database must be granted REMOTE privilege.

The single exception is the database immediately above the current database in a SQL Remote hierarchy, which must be granted CONSOLIDATE privilege.

The remote user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes.

For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

The GRANT REMOTE statement is required for the remote database to receive messages, but does not by itself subscribe the remote user to any data. To subscribe to data, a subscription must be created for the user ID to one of the publications in the current database, using the database extraction utility or the CREATE SUBSCRIPTION statement.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted REMOTE privilege without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. To run the Message Agent continuously, you must ensure that every user with REMOTE privilege has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many consolidated databases, the Message Agent is run continuously, so that all remote databases would have a SEND clause specified. A typical setup may involve sending messages to laptop users daily (SEND AT) and to remote servers every hour or two (SEND EVERY). You should use as few different times as possible, for efficiency.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement grants remote privilege to user Sam_Singer, using an SMTP email system, sending messages to the address Singer, Samuel once every two hours:

```
GRANT REMOTE TO Sam_Singer
TYPE SMTP
ADDRESS 'Singer, Samuel'
SEND EVERY '02:00';
```

Related Information

REMOTE Privilege
Subscriptions
Send Frequency
SQL Remote Message Systems
Granting REMOTE Privilege (SQL Central)
REVOKE REMOTE Statement [SQL Remote] [page 1351]
GRANT PUBLISH Statement [SQL Remote] [page 1206]
GRANT CONSOLIDATE Statement [SQL Remote] [page 1199]
CREATE SUBSCRIPTION Statement [SQL Remote] [page 993]
CREATE REMOTE [MESSAGE] Statement [SQL Remote] [page 950]

1.4.4.186 GRANT ROLE Statement

Grant roles to users and roles.

Syntax

Grant system roles

```
GRANT ROLE system-role
TO grantee [ ,... ]
```

```
system-role :
dbo
| DIAGNOSTICS
| PUBLIC
| rs_systabgroup
| SA_DEBUG
| SYS
| SYS_REPLICATION_ADMIN_ROLE
| SYS_RUN_REPLICATION_ROLE
| SYS_SAMONITOR_ADMIN_ROLE
| SYS_SPATIAL_ADMIN_ROLE
```

Grant user-defined roles

```
GRANT ROLE user-defined-role [,...]
```

```
TO grantee [ ,... ]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

Grant compatibility roles

```
GRANT ROLE compatibility-role-name [,...]
TO grantee [ ,... ]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
[ WITH NO SYSTEM PRIVILEGE INHERITANCE ]
```

```
compatibility-role-name :
SYS_AUTH_BACKUP_ROLE
| SYS_AUTH_DBA_ROLE
| SYS_AUTH_PROFILE_ROLE
| SYS_AUTH_READCLIENTFILE_ROLE
| SYS_AUTH_READFILE_ROLE
| SYS_AUTH_RESOURCE_ROLE
| SYS_AUTH_SA_ROLE
| SYS_AUTH_SSO_ROLE
| SYS_AUTH_VALIDATE_ROLE
| SYS_AUTH_WRITECLIENTFILE_ROLE
| SYS_AUTH_WRITEFILE_ROLE
```

Parameters

role-name

The name of a system role, compatibility role, user-extended role, or user-defined role.

grantee

The user ID of a user, or the name of a role. You cannot grant roles to compatibility roles. You can grant roles to any system role; however, only the following system roles support logins: PUBLIC, dbo, diagnostics, rs_systabgroup, and SA_DEBUG

WITH [NO] ADMIN OPTION clause

This clause is only applicable when granting non-system roles. You cannot grant administrative rights on a system role; only users with the MANAGE ROLES system privilege can administer (grant and revoke) system roles.

The default is WITH NO ADMIN OPTION, meaning that the `grantee` is given the role, but not the ability to administer it.

If WITH ADMIN OPTION is specified, each `grantee` is given administrative rights over each `role-granted`.

If WITH ADMIN ONLY OPTION is specified, then the `grantee` is given only administrative rights over the role, but is not given the role itself. You can never use the WITH NO SYSTEM PRIVILEGE INHERITANCE clause with the WITH ADMIN ONLY OPTION.

You can only use the WITH ADMIN OPTION clause with the WITH NO SYSTEM PRIVILEGE INHERITANCE clause when granting SYS_AUTH_DBA_ROLE.

WITH NO SYSTEM PRIVILEGE INHERITANCE

This clause prevents the grantees of a role from inheriting the role's system privileges. Normally, when you grant a compatibility role to a user or role, the compatibility role's system privileges are available to both

the role and its grantees. When you disable the inheritance of a compatibility role's system privileges, the system privileges are available only to the role, not to its grantees.

The WITH NO SYSTEM PRIVILEGE INHERITANCE clause is provided for backwards compatibility. Disabling system privilege inheritance for a compatibility role mimics the behavior of the non-inheritable authority in version 12 and earlier databases. Enabling system privilege inheritance for a compatibility role mimics the behavior of all system roles and user-defined roles.

You can disable the inheritance of the system privileges when granting one of the following roles to users, user-extended roles, or system roles:

- SYS_AUTH_DBA_ROLE role
- SYS_AUTH_RESOURCE_ROLE
- SYS_AUTH_BACKUP_ROLE
- SYS_AUTH_VALIDATE_ROLE
- SYS_RUN_REPLICATION_ROLE

Also, you can only use the WITH ADMIN OPTION clause with the WITH NO SYSTEM PRIVILEGE INHERITANCE clause when granting SYS_AUTH_DBA_ROLE. You can never use the WITH NO SYSTEM PRIVILEGE INHERITANCE clause with the WITH ADMIN ONLY OPTION.

Disabling the system privilege inheritance for a user is only useful if you intend to convert the user to a user-extended role.

Remarks

With the exception of the SYS role, you can grant/revoke additional roles to/from a system role, provided you have administrative rights on the roles you are granting/revoking.

When granting a role to the MANAGE ROLES system privilege, you must use the special internal representation SYS_MANAGE_ROLES_ROLE. For example, `GRANT ROLE role-name TO SYS_MANAGE_ROLES_ROLE;`.

The GRANT syntax related to authorities, permissions, and groups used in pre-16.0 versions of the software is still supported but deprecated.

Privileges

You must have administrative rights for each role you grant.

To grant the SYS_REPLICATION_ADMIN_ROLE system role, you must have the MANAGE ROLES system privilege.

To grant the SYS_REPLICATION_RUN_ROLE system role, you must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Granting roles to users

To grant user Bob the role called SecurityRole without administrative rights, execute the following statement:

```
GRANT ROLE SecurityRole TO Bob;
```

To grant the role RoleB all the privileges associated with RoleA but no administrative rights for RoleA, execute the following statement:

```
GRANT ROLE RoleA TO RoleB;
```

To grant RoleB along with its administrative rights to the user Jane, execute the following statement:

```
GRANT ROLE RoleB TO Jane WITH ADMIN OPTION;
```

To grant the user John the administrative rights to RoleB, but the inability to use RoleB, execute the following statement:

```
GRANT ROLE RoleB TO John WITH ADMIN ONLY OPTION;
```

Granting the SYS_RUN_REPLICATION_ROLE system role

The following example grants the SYS_RUN_REPLICATION_ROLE system role to grantee Sam_Singer:

```
GRANT ROLE SYS_RUN_REPLICATION_ROLE
TO Sam_Singer;
```

Granting the SYS_REPLICATION_ADMIN_ROLE system role

The following example grants the SYS_REPLICATION_ADMIN_ROLE role to grantee Sam_Singer:

```
GRANT ROLE SYS_REPLICATION_ADMIN_ROLE
TO Sam_Singer;
```

Related Information

Role Administrators
Initiation of Synchronization
System Privileges
Roles
Compatibility Roles
User-defined Roles
Impersonation
Replication-related System Roles
Changes in Inheritance Behavior for Some Authorities That Became Roles
SETUSER Statement [page 1403]
GRANT Statement (Authorities and Groups) (Deprecated)
REVOKE ROLE Statement [page 1352]

1.4.4.187 GRANT USAGE ON SEQUENCE Statement

Grants privilege to use a specified sequence.

Syntax

```
GRANT USAGE ON SEQUENCE sequence-name
TO userid, ...
```

Remarks

This privilege allows the user to select the current or next value from the specified sequence.

Privileges

You must have the MANAGE ANY OBJECT PRIVILEGE privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Part of optional ANSI/ISO SQL Language Feature T176.

Example

A ticketing application creates a sequence to ensure that each ticket has an unique number.

```
CREATE SEQUENCE TicketNumber
   MINVALUE 1000
   MAXVALUE 100000;
```

A user, TicketAgent, using this application to generate tickets must be granted the usage rights to the sequence so that they can select values from the sequence.

```
GRANT USAGE ON SEQUENCE TicketNumber to TicketAgent;
```

Now, TicketAgent can select values from the TicketNumber sequence and use them to create new tickets. For example:

```
INSERT INTO NewTicket
   SELECT TicketNumber.nextval, 'Concert Ticket', ...;
```

Related Information

Use of a Sequence to Generate Unique Values

1.4.4.188 GROUP BY Clause

Groups columns, alias names, and functions as part of the SELECT statement.

Syntax

```
GROUP BY
| group-by-term, ...
| simple-group-by-term, ...  WITH ROLLUP
| simple-group-by-term, ... WITH CUBE
| GROUPING SETS ( group-by-term, ... )
```

```
group-by-term :
simple-group-by-term
| ( simple-group-by-term, ... )
| ROLLUP ( simple-group-by-term, ... )
| CUBE ( simple-group-by-term, ... )
```

```
simple-group-by-term :
expression
| ( expression )
| ()
```

Parameters

GROUPING SETS clause

The GROUPING SETS clause allows you to perform aggregate operations on multiple groupings from a single query specification. Each set specified in a GROUPING SET clause is equivalent to a GROUP BY clause.

For example, the following two queries are equivalent:

```
SELECT a, b, SUM( c ) FROM t
GROUP BY GROUPING SETS ( ( a, b ), ( a ), ( b ), ( ) );
```

```
SELECT a, b, SUM( c ) FROM t
  GROUP BY a, b
UNION ALL
SELECT a, NULL, SUM( c ) FROM t
  GROUP BY a
UNION ALL
SELECT NULL, b, SUM( c ) FROM t
  GROUP BY b
UNION ALL
SELECT NULL, NULL, SUM( c ) FROM t;
```

A grouping expression may be reflected in the result set as a NULL value, depending on the grouping in which the result row belongs. This may cause confusion over whether the NULL is the result of another grouping, or whether the NULL is the result of an actual NULL value in the underlying data. To distinguish between NULL values present in the input data and NULL values inserted by the grouping operator, use the GROUPING function.

Specifying an empty set of parentheses ( ) in the GROUPING SETS clause returns a single row containing the overall aggregate.

ROLLUP clause

The ROLLUP clause is similar to the GROUPING SETS clause in that it can be used to specify multiple grouping specifications within a single query specification. A ROLLUP clause of `n simple-group-by-term`s generates `n`+1 grouping sets, formed by starting with the empty parentheses, and then appending successive `group-by-term`s from left to right.

For example, the following two statements are equivalent:

```
SELECT a, b, SUM( c ) FROM t
GROUP BY ROLLUP ( a, b );
```

```
SELECT a, b, SUM( c ) FROM t
GROUP BY GROUPING SETS ( ( a, b ), a, ( ) );
```

You can use a ROLLUP clause within a GROUPING SETS clause.

CUBE clause

The CUBE clause is similar to the ROLLUP and GROUPING SETS clauses in that it can be used to specify multiple grouping specifications within a single query specification. The CUBE clause is used to represent all possible combinations that can be made from the expressions listed in the CUBE clause.

For example, the following two statements are equivalent:

```
SELECT a, b, SUM( c ) FROM t
GROUP BY CUBE ( a, b, c );
```

```
SELECT a, b, SUM( c ) FROM t
GROUP BY GROUPING SETS ( ( a, b, c ), ( a, b ), ( a, c ),
 ( b, c ), a, b, c, () );
```

You can use a CUBE clause within a GROUPING SETS clause.

WITH ROLLUP clause

This is an alternative syntax to the ROLLUP clause, and is provided for Transact-SQL compatibility.

WITH CUBE clause

This is an alternate syntax to the CUBE clause, and is provided for Transact-SQL compatibility.

Remarks

When using the GROUP BY clause, you can group by expressions (with some limitations), columns, alias names, or functions. The result of the query contains one row for each distinct value (or set of values) of each grouping set.

The empty GROUP BY list, (), signifies the treatment of the entire input as a single group. For example, the following two statements are equivalent:

```
SELECT COUNT(), SUM(Salary) FROM GROUPO.Employees;
```

```
SELECT COUNT(), SUM(Salary) FROM GROUPO.Employees GROUP BY ();
```

Privileges

None.

Standards

ANSI/ISO SQL Standard

Core Feature. GROUPING SETS, GROUP BY (), ROLLUP, and CUBE constitute portions of optional ANSI/ISO SQL Language Feature T431. The software does not support optional ANSI/ISO SQL Language Feature T432, "Nested and concatenated GROUPING SETS".

Extensions to the GROUP BY clause that are not in the standard include:

- WITH ROLLUP
- WITH CUBE
- the ability to specify arbitrary expressions as GROUP BY terms. In the ANSI/ISO SQL Standard, every GROUP BY term must be a column reference from an underlying table in the query's FROM clause.

Example

The following example returns a result set showing the total number of orders, and then provides subtotals for the number of orders in each year (2000 and 2001).

```
SELECT YEAR( OrderDate ) Year, QUARTER( OrderDate ) Quarter, count(*) Orders
FROM GROUPO.SalesOrders
GROUP BY ROLLUP ( Year, Quarter )
ORDER BY Year, Quarter;
```

Like the preceding ROLLUP operation example, the following CUBE query example returns a result set showing the total number of orders and provides subtotals for the number of orders in each year (2000 and 2001). Unlike ROLLUP, this query also gives subtotals for the number of orders in each quarter (1, 2, 3, and 4).

```
SELECT YEAR(OrderDate) Year, QUARTER( OrderDate ) Quarter, count(*) Orders
FROM GROUPO.SalesOrders
GROUP BY CUBE ( Year, Quarter )
ORDER BY Year, Quarter;
```

The following example returns a result set that gives subtotals for the number of orders in the years 2000 and 2001. The GROUPING SETS operation lets you select the columns to be subtotaled instead of returning all combinations of subtotals like the CUBE operation.

```
SELECT YEAR(OrderDate) Year, QUARTER( OrderDate ) Quarter, count(*) Orders
FROM GROUPO.SalesOrders
GROUP BY GROUPING SETS ( ( Year, Quarter ), ( Year ) )
ORDER BY Year, Quarter;
```

Related Information

Summarizing, Grouping, and Sorting Query Results
GROUP BY Clause Extensions
GROUP BY and the SQL/2008 Standard
GROUP BY GROUPING SETS
The CUBE Clause
The ROLLUP Clause
SELECT Statement [page 1362]
GROUPING Function [Aggregate] [page 396]

1.4.4.189 HELP Statement [Interactive SQL]

Provides help in the Interactive SQL environment.

Syntax

```
HELP [ 'topic' ]
```

Remarks

The HELP statement is used to access SQL Anywhere documentation.

The `topic` for help can be optionally specified. You must enclose `topic` in single quotes. In some help formats, the topic cannot be specified; in this case, a link to a general help page for Interactive SQL appears.

You can specify the following `topic` values:

- SQL Anywhere error codes
- SQL statement keywords (such as INSERT, UPDATE, SELECT, CREATE DATABASE)

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Interactive SQL

1.4.4.190 IF Statement

Controls conditional execution of SQL statements.

Syntax

```
IF search-condition THEN statement-list
[ ELSEIF { search-condition | operation-type } THEN statement-list ] ...
[ ELSE statement-list ]
{ END IF | ENDIF }
```

Remarks

The IF statement is a control statement that allows you to conditionally execute the first list of SQL statements whose `search-condition` evaluates to TRUE. If no `search-condition` evaluates to TRUE, and an ELSE clause exists, the `statement-list` in the ELSE clause is executed.

Execution resumes at the first statement after the END IF.

Note

Do not confuse the syntax of the IF statement with that of the IF expression.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The IF statement is part of optional ANSI/ISO SQL Language Feature P002, "Computational completeness". The ENDIF keyword is not in the standard.

Example

The following procedure illustrates the use of the IF statement:

```
CREATE PROCEDURE TopCustomer2 (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
   DECLARE err_notfound EXCEPTION
   FOR SQLSTATE '02000';
   DECLARE curThisCust CURSOR FOR
   SELECT CompanyName, CAST(    sum(SalesOrderItems.Quantity *
   Products.UnitPrice) AS INTEGER) VALUE
   FROM Customers
   LEFT OUTER JOIN SalesOrders
   LEFT OUTER JOIN SalesOrderItems
   LEFT OUTER JOIN Products
   GROUP BY CompanyName;
   DECLARE ThisValue INT;
   DECLARE ThisCompany CHAR(35);
   SET TopValue = 0;
   OPEN curThisCust;
   CustomerLoop:
   LOOP
      FETCH NEXT curThisCust
      INTO ThisCompany, ThisValue;
      IF SQLSTATE = err_notfound THEN
         LEAVE CustomerLoop;
      END IF;
      IF ThisValue > TopValue THEN
         SET TopValue = ThisValue;
         SET TopCompany = ThisCompany;
      END IF;
   END LOOP CustomerLoop;
   CLOSE curThisCust;
END;
```

Related Information

Stored Procedures, Triggers, Batches, and User-defined Functions
BEGIN Statement [page 784]
IF Expressions [page 38]
Search Conditions [page 58]

1.4.4.191 IF Statement [T-SQL]

Controls conditional execution of a SQL statement, as an alternative to the Watcom SQL IF statement.

Syntax

```
 IF expression statement
[ ELSE [ IF expression ] statement ]
```

Remarks

The Transact-SQL IF conditional and the ELSE conditional each control the execution of only a single SQL statement or compound statement (between the keywords BEGIN and END).

In comparison to the Watcom SQL IF statement, there is no THEN in the Transact-SQL IF statement. The Transact-SQL version also has no ELSEIF or END IF keywords.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example illustrates the use of the Transact-SQL IF statement:

```
IF (SELECT max(ID) FROM sysobjects) < 100
   RETURN
ELSE
   BEGIN
      PRINT 'These are the user-created objects'
      SELECT name, type, ID
      FROM sysobjects
      WHERE ID < 100
   END
```

The following two statement blocks illustrate Transact-SQL and Watcom SQL compatibility:

```
/* Transact-SQL IF statement */
IF @v1 = 0
   PRINT '0'
ELSE IF @v1 = 1
   PRINT '1'
ELSE
   PRINT 'other'
```

```
/* Watcom SQL IF statement */
IF v1 = 0 THEN
   PRINT '0'
ELSEIF v1 = 1 THEN
   PRINT '1'
ELSE
   PRINT 'other'
END IF
```

1.4.4.192 INCLUDE Statement [ESQL]

Includes a file into a source program to be scanned by the SQL preprocessor.

Syntax

```
INCLUDE filename
```

```
filename : SQLDA | SQLCA | string
```

Remarks

The INCLUDE statement is very much like the C preprocessor #include directive. The SQL preprocessor reads an Embedded SQL source file and replaces all the Embedded SQL statements with C-language source code. If a file contains information that the SQL preprocessor requires, include it with the Embedded SQL INCLUDE statement.

Two file names are specially recognized: SQLCA and SQLDA. The following statement must appear before any Embedded SQL statements in all Embedded SQL source files.

```
EXEC SQL INCLUDE SQLCA;
```

This statement must appear at a position in the C program where static variable declarations are allowed. Many Embedded SQL statements require variables (invisible to the application developer), which are declared by the SQL preprocessor at the position of the SQLCA include statement. The SQLDA file must be included if any SQLDAs are used.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

1.4.4.193 INPUT Statement [Interactive SQL]

Imports data into a database table from an external file, from the keyboard, from an ODBC data source, or from a shapefile.

Syntax

Syntax:

Import from an external file or from the keyboard

```
INPUT INTO [ owner.]table-name input-options
```

```
input-options :
[ ( column-name, ... ) ]
[ BYTE ORDER MARK { ON | OFF } ]
[ COLUMN WIDTHS ( integer, ...) ]
[ DELIMITED BY string ]
[ ENCODING encoding ]
[ ESCAPE CHARACTER character ]
[ ESCAPES { ON | OFF } ]
[ FORMAT input-format ]
[ FROM filename | PROMPT ]
[ NOSTRIP ]
[ SKIP integer ]
```

```
input-format :
TEXT
| EXCEL
| FIXED
| SHAPEFILE
```

```
encoding : identifier | string
```

Syntax:

Import from an ODBC data source

```
INPUT USING connection-string
FROM source-table-name
INTO destination-table-name
[ CREATE TABLE { ON | OFF } ]
```

```
connection-string :
{ DRIVER=odbc-driver-name
| DSN=odbc-data-source } [ ; { connection-parameter = value } ]
```

Syntax:

Import from a shapefile

```
INPUT INTO [ owner.]table-name
FROM filename
FORMAT SHAPEFILE
[ SRID srid-number ]
[ ENCODING encoding ]
```

```
encoding : identifier | string
```

Parameters

column-name

List the names of the columns in the table in the order that corresponds to the order of the values in the input file. Use this parameter when the order of the table columns is different from the order of the values in the input file, or there are fewer columns in the input file than there are in the table.

BYTE ORDER MARK clause

Use this clause to specify whether to process a byte order mark (BOM) in the input file.

The BYTE ORDER MARK clause is relevant only when reading from TEXT formatted files. Attempts to use the BYTE ORDER MARK clause with FORMAT clauses other than TEXT causes an error.

The BYTE ORDER MARK clause is used only when reading or writing files encoded with UTF-8 or UTF-16 (and their variants). Attempts to use the BYTE ORDER MARK clause with any other encoding causes an error.

If the ENCODING clause is specified:

- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, Interactive SQL searches for a BOM at the beginning of the data. If a BOM is present, it is used to verify the endianness of the data. If you specify the wrong endian, an error is returned.
- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, Interactive SQL searches for a BOM at the beginning of the data. If a BOM is present, it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.
- If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, Interactive SQL searches for a BOM at the beginning of the data. If a BOM is present it is ignored.

If the ENCODING clause is not specified:

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, Interactive SQL looks for a BOM at the beginning of the input data. If a BOM is located, the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or UTF-8) and the BOM is not considered to be part of the data to be loaded.
- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, or a BOM is not found at the beginning of the input data, encoding on the client computer is used.

COLUMN WIDTHS clause

The COLUMN WIDTHS clause specifies the widths of the columns in the input file and can only be specified for FIXED format. If COLUMN WIDTHS is not specified, the widths are determined by the database column types.

CREATE TABLE clause

Use the CREATE TABLE clause to specify whether to create the destination table if it does not exist. The default is ON. The CREATE TABLE ON clause can only be used when importing from an ODBC data source.

DELIMITED BY clause

The DELIMITED BY clause is only supported for the TEXT input format. This clause allows you to specify a string to be used as the delimiter between column values in the TEXT input format.

The default delimiter is a comma for locales that use a period as the decimal separator, and a semicolon for locales that use commas as the decimal separator.

ENCODING clause

The `encoding` argument specifies the encoding that is used to read the file. The ENCODING clause can only be used with the TEXT and SHAPEFILE formats.

When running Interactive SQL, the encoding that is used to import the data is determined in the following order:

- The encoding specified by the ENCODING clause (if this clause is specified).
- The encoding specified by the default_isql_encoding option (if this option is set).
- If the file has a byte-order mark, the appropriate Unicode encoding is used.
- The default encoding for the computer you are running on. On English Windows computers, the default encoding is 1252 or 850.

If the input file was created using the OUTPUT statement and an encoding was specified, then the same ENCODING clause should be specified on the INPUT statement.

ESCAPE CHARACTER clause

The default escape character for hexadecimal codes is a backslash (\). For example, \x0A is the line feed character.

The newline character can be specified as \n. Other characters can be specified using hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\ ) is interpreted as a single backslash. A backslash followed by any character other than n, x, X, or \ is interpreted as two separate characters. For example, \q is interpreted as a backslash and the letter q.

The escape character can be changed using the ESCAPE CHARACTER clause. For example, to use the exclamation mark as the escape character, specify:

```
... ESCAPE CHARACTER '!'
```

ESCAPES clause

With ESCAPES turned on (the default), characters following the escape character are interpreted as special characters by the database server. With ESCAPES turned off, the characters are read exactly as they appear in the source.

FORMAT clause

The FORMAT clause allows you to specify the file format for the input. If you do not specify the FORMAT clause, the format specified by the input_format option is used. If you specify the FORMAT clause, the setting of the input_format option is ignored. The default input format is TEXT. Allowable input formats are:

TEXT

Input lines are assumed to be characters, one row per line, with column values separated by delimiters. Alphabetic strings may be enclosed in single quotes or double quotes. Strings containing delimiters must be enclosed in either single or double quotes. If the string itself contains single or double quotes, double the quote character to use it within the string. Use the DELIMITED BY clause to specify a field separator.

Three other special sequences are also recognized. The two characters \n represent a newline character, \\ represents a single (\), and the sequence \xDD represents the character with hexadecimal code DD.

Omitted values are treated as NULL. If the value in that position cannot be NULL, a zero is inserted in numeric columns and an empty string in character columns.

EXCEL

When files with a `.txt` or `.csv` extension are imported with the FORMAT EXCEL clause, they follow the default formatting for Microsoft Excel workbook files.

The default worksheet used is the one whose name comes first in alphabetical order. To specify a specific worksheet, use the WORKSHEET clause. The first row in the worksheet is assumed to contain the column headings.

Input from a workbook file (`.xl*`) is only supported on Windows.

FIXED

Input lines are in fixed format. The width of the columns can be specified using the COLUMN WIDTHS clause. If they are not specified, column widths in the file must be the same as the maximum number of characters allowed by the corresponding database column.

The FIXED format cannot be used with binary columns that contain embedded newline and end-of-file character sequences.

SHAPEFILE

Input is in the form of an ESRI shapefile. Unlike the LOAD statement, when loading shapefiles using the INPUT statement, the shapefile must be located on the client computer. The associated `.shx` and `.dbf` files must be located in the same directory as the `.shp` file, and have the same base file name.

If an encoding is not specified when loading a shapefile, the default is ISO-8859-1.

Use the SRID clause to specify a SRID to associate with the geometries. If you do not specify a SRID, SRID 0 is used by default. Ideally, you should specify the same SRID as the one mentioned in the project file (`.prj`) for the shapefile. If that SRID is not available, use one that is equivalent. SQL Anywhere provides thousands of SRIDs you can add to the database using the sa_install_feature system procedure.

To use other formats such as, DBASE II, DBASE III, Microsoft Visual FoxPro, Lotus 123, or Microsoft Excel 97, use INPUT USING.

FROM filename clause

The `filename` can be quoted or unquoted. If the string is quoted, it is subject to the same formatting requirements as other SQL strings.

To indicate directory paths, the backslash character (\) must be represented by two backslashes when the path is quoted.

The location of a relative `filename` is determined as follows:

- If the INPUT statement is executed directly in Interactive SQL, the path to `filename` is resolved relative to the directory in which Interactive SQL is running. For example, suppose you open Interactive SQL from the directory `c:\work` and execute the following statement:

  ```
  INPUT INTO GROUPO.Employees
   FROM 'inputs\\inputfile.dat';
  ```

  Interactive SQL looks for `c:\work\inputs\inputfile.dat`.
- If the INPUT statement resides in a `.sql` file, Interactive SQL first attempts to resolve the path to `filename` relative to the location of the file. If unsuccessful, Interactive SQL looks for `filename` in a path relative to the directory in which Interactive SQL is running.
  For example, suppose you had a file, `c:\homework\inputs.sql`, that contained the following statement:

  ```
  INPUT INTO GROUPO.Employees
   FROM 'inputs\\inputfile.dat';
  ```

  Interactive SQL would first look for `inputfile.dat` in `c:\homework\inputs`. If Interactive SQL does not find `inputfile.dat` in that location, Interactive SQL looks in the directory in which Interactive SQL is running.

FROM source-table-name clause

The `source-table-name` parameter is a quoted string containing the name of the table in the source database. The name can be in the form `database-name.owner.table-name`, `owner.table-name`, or `table-name`. Use a period to separate the components, even if that is not the native separator in the source database. If the source database requires a database name, but not an owner name, the format of `source-table-name` must be `database..table` (in this case the owner name is empty). Do not quote any of the names in the parameter. For example, do not use `'dba."my-table"'`; use `'dba.my-table'` instead.

INTO clause

The name of the table to input the data into.

PROMPT clause

The PROMPT clause allows the user to enter values for each column in a row. When running in windowed mode, a window is displayed, allowing the user to enter the values for the new row. If you are running Interactive SQL from the command line, Interactive SQL prompts you to type the value for each column on the command line.

NOSTRIP clause

Normally, for TEXT input format, trailing blanks are stripped from unquoted strings before the value is inserted. NOSTRIP can be used to suppress trailing blank stripping. Trailing blanks are not stripped from quoted strings, regardless of whether the option is used. Leading blanks are stripped from unquoted strings, regardless of the NOSTRIP option setting.

SKIP clause

When inserting lines from a TEXT file, the SKIP clause omits the specified number of lines starting at the beginning of the file. The number specified must be a non-negative integer. The SKIP clause is for TEXT format only.

If the specified number of lines exceeds the number of lines in the file, the INPUT statement inserts no data and no error is returned.

USING clause

The USING clause inputs data from an ODBC data source. You can either specify the ODBC data source name with the DSN option, or the ODBC driver name and connection parameters with the DRIVER option. The `connection-parameter` parameter is a list of database-specific connection parameters.

The `odbc-data-source` parameter is the name of a user or ODBC data source name. For example, `odbc-data-source` for the SQL Anywhere sample database is SQL Anywhere 17 Demo.

The `odbc-driver-name` parameter is the ODBC driver name. For a SQL Anywhere database, `odbc-driver-name` is SQL Anywhere 17. For an UltraLite database, `odbc-driver-name` is UltraLite 17.

WORKSHEET clause

The WORKSHEET clause identifies the name of the worksheet within the Microsoft Excel file that data is imported from. If the clause is omitted, then the first sheet in the Microsoft Excel file is used.

Remarks

The INPUT statement allows efficient mass insertion into a named database table. Lines of input are read either from the user via an input window (if PROMPT is specified) or from a file (if FROM `filename` is specified). If neither is specified, the input is read from the SQL script file that contains the INPUT statement. In Interactive SQL, this can even be directly read from the *SQL Statements* pane. In this case, input is ended with a line containing only the string END.

When the input is read directly from the *SQL Statements* pane, you must specify a semicolon before the values for the records to be inserted at the end of the INPUT statement. For example:

```
INPUT INTO Owner.TableName;
value1, value2, value3
value1, value2, value3
value1, value2, value3
value1, value2, value3
END
```

The END keyword (not a semicolon) terminates data for INPUT statements that do not name a file and do not include the PROMPT keyword.

If a column list is specified, the data is inserted into the specified columns of the named table. By default, the INPUT statement assumes that column values in the input file appear in the same order as they appear in the database table definition. If the table column order is different, you must use `column-name` parameter to list the table columns in the same order as the column values in the input file.

By default, the INPUT statement stops when it attempts to insert a row that causes an error. Errors can be treated in different ways by setting the on_error and conversion_error options.

Interactive SQL displays a warning on the *History* tab if any string values are truncated on INPUT. Missing values for NOT NULL columns are set to zero for numeric types and to the empty string for non-numeric types. If INPUT attempts to insert a NULL row, the input file contains an empty row.

Because the INPUT statement is an Interactive SQL statement, you cannot use it in any compound statement (such as an IF statement), in a stored procedure, or in any statement executed by the database server.

Privileges

You must be the owner of the table, or have the INSERT ANY TABLE system privilege, or have INSERT privilege on the table. You must also have the SELECT ANY TABLE system privilege, or have SELECT privilege on the table.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example imports data into the Employees table from a fictitious TEXT file, `new_emp.inp`:

```
INPUT INTO GROUPO.Employees
   FROM new_emp.inp
   FORMAT TEXT;
```

This example copies the table ulTest, into a table called saTest. The ulTest table is in an UltraLite database in the file `C:\test\myULDatabase.udb`, and the saTest table is created in *demo.db*:

```
INPUT USING 'driver=UltraLite 17;dbf=C:\\test\\myULDatabase.udb'
   FROM "ulTest" INTO "saTest";
```

This example loads the shapefile `myshapefile.shp` into the myTable table, and assigns SRID 4269 to the geometries in the shapefile.

```
INPUT INTO myTable
   FROM 'myshapefile.shp'
   FORMAT SHAPEFILE SRID 4269
```

This example adds a new row to the Products table, and prompts the user to enter the values for each column.

```
INPUT INTO GROUPO.Products PROMPT;
```

This example loads data from the file `c:\temp\input.dat` into the Employees table. Note how the backslashes are doubled.

```
INPUT INTO GROUPO.Employees
FROM 'c:\\temp\\input.dat';
```

The following example creates a table, myInventory, and imports data from the file `stock.txt` that contains the data but in a different column order than the table definition. To correct the order mismatch, the correct column order required for the import is specified by the `column-name` parameter after the table name in the INPUT statement. That is, **`(item, Quantity)`** instructs Interactive SQL to take the first column value in the input file and place it in the second column of the table, and then take the second column value in the input file and place it in the first column of the table.

```
CREATE TABLE myInventory (
Quantity INTEGER,
item VARCHAR(60)
);
INPUT INTO myInventory (item, Quantity)
FROM stock.txt;
```

The following example loads data from the first worksheet in file `c:\test\Book1.xlsx` into the existing table named inputTest:

```
INPUT INTO inputTest FROM 'c:\\test\\Book1.xlsx' FORMAT EXCEL
```

Related Information

Data Import with the INPUT Statement
Data Import
Interactive SQL
Supported Character Sets
Statements Allowed in Procedures, Triggers, Events, and Batches
Inserting a New ow into an Interactive SQL Result Set
Tutorial: Experimenting with the Spatial Features
OUTPUT Statement [Interactive SQL] [page 1289]
INSERT Statement [page 1232]
SET OPTION Statement [Interactive SQL] [page 1391]
LOAD TABLE Statement [page 1247]
Unload Utility (dbunload)
Support for ESRI Shapefiles
input_format Option [Interactive SQL]
sa_install_feature System Procedure [page 1625]
default_isql_encoding Option [Interactive SQL]
isql_field_separator Option [Interactive SQL]

1.4.4.194 INSERT Statement

Inserts a single row or a selection of rows from elsewhere in the database into a table.

Syntax

Insert one or more rows

```
INSERT [ INTO ] [ owner.]table-name [ ( column-name, ... ) ]
[ ON EXISTING {
    ERROR
    | SKIP
    | UPDATE [ DEFAULTS { ON | OFF } ]
    } ]
{ DEFAULT VALUES
    | VALUES row-value-constructor }
[ OPTION( query-hint[, ... ] ) ]
```

Insert the results of a SELECT statement

```
INSERT [ INTO ] [ owner.]table-name [ ( [ column-name [, ... ] ] ) ]
[ ON EXISTING {
    ERROR
    | SKIP
    | UPDATE [ DEFAULTS { ON | OFF } ]
    } ]
[ WITH AUTO NAME ]
select-statement
[ OPTION( query-hint[, ... ] ) ]
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| FORCE NO OPTIMIZATION
| option-name = option-value
```

```
option-name :
 identifier
```

```
option-value :
 hostvar (indicator allowed)
| string
| identifier
| number
```

```
insert-expression :
 expression | DEFAULT
```

```
row-value-constructor :
 ( [ insert-expression [, ... ] ] ) [,( [ insert-expression [, ... ] ] ) ... ]
```

Parameters

VALUES clause

Use the VALUES clause to specify the values to insert. To insert the default values defined for the columns, specify DEFAULT VALUES. You can also specify VALUES ( ), which is equivalent to DEFAULT VALUES. The VALUES clause also support row value constructors so that you can insert multiple rows of values in a single statement. The number and order of `insert-expression` values in each `row-value-constructor` must correspond to the column list specified in the INTO clause. If a column list is not specified, it is assumed to be the complete ordered column list for the table. If you specify an empty column list (), then each of the columns in the table must have a default value. If you specify multiple row value constructors, they must be separated by commas. If the `insert-expression` is a host variable or procedure variable, then an error is returned. Using single-row INSERT...VALUES for host and procedure variables is supported. If an error occurs while inserting any of the rows, all of the changes are rolled back.

WITH AUTO NAME clause

WITH AUTO NAME applies only to syntax 2. If you specify WITH AUTO NAME, the names of the items in the query block determine which column the data belongs in. The query block items should be either column references or aliased expressions. Destination columns not defined in the query block are assigned their default value. This is useful when the number of columns in the destination table is very large.

The INSERT statement returns an error if the WITH AUTO NAME clause is specified and the query block contains columns that do not match columns in the target table. For example, executing the following statement returns an error indicating that the operation column in the SELECT query block cannot be found in the table:

```
CREATE TABLE MyTable5(
      pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
      TableName CHAR(128),
      TableNameLen INT );
INSERT
INTO MyTable5 WITH AUTO NAME
SELECT length(t.table_name) AS TableNameLen,
      t.table_name AS TableName, 1 as operation
FROM SYS.SYSTAB t
WHERE table_id <= 10;
```

ON EXISTING clause

The ON EXISTING clause of the INSERT statement applies to both syntaxes. It updates existing rows in a table, based on primary key lookup, with new column values. This clause can only be used on tables that have a primary key. Attempting to use this clause on tables without primary keys generates a syntax error. You cannot insert values into a proxy table with the ON EXISTING clause.

> **i** Note
>
> If you anticipate many rows qualifying for the ON EXISTING condition, consider using the MERGE statement instead. The MERGE statement provides more control over the actions you can take for matching rows. It also provides a more sophisticated syntax for defining what constitutes a match.

If you specify the ON EXISTING clause, the database server performs a primary key lookup for each input row. If the corresponding row does not already exist in the table, it inserts the new row. For rows that already exist in the table, you can choose to silently ignore the input row (SKIP), generate an error message for duplicate key values (ERROR), or update the old values using the values from the input row (UPDATE). By default, if you do not specify the ON EXISTING clause, attempting to insert rows into a table where the row already exists results in a duplicate key value error, and is equivalent to specifying the ON EXISTING ERROR clause. Rows that are skipped are included in the @@rowcount variable.

When using the ON EXISTING UPDATE clause, the input row is compared to the stored row. Any column values explicitly stated in the input row replace the corresponding column values in the stored row.

Likewise, column values not explicitly stated in the input row result in no change to the corresponding column values in the stored row, with the exception of columns with defaults. When using the ON EXISTING UPDATE clause with columns that have defaults (including DEFAULT AUTOINCREMENT columns), you can further specify whether to update the column value with the default values by specifying ON EXISTING UPDATE DEFAULTS ON, or leave the column value as it is by specifying ON EXISTING UPDATE DEFAULTS OFF. If nothing is specified, the default behavior is ON EXISTING UPDATE DEFAULTS OFF.

> **i Note**
>
> DEFAULTS ON and DEFAULTS OFF parameters do not affect values in DEFAULT TIMESTAMP, DEFAULT UTC TIMESTAMP, or DEFAULT LAST USER. For these columns, the value in the stored row is always updated during the UPDATE.

When using the ON EXISTING SKIP and ON EXISTING ERROR clauses, if the table contains default columns, the server computes the default values even for rows that already exist. As a result, default values such as AUTOINCREMENT cause side effects even for skipped rows. In this case of AUTOINCREMENT, this results in skipped values in the AUTOINCREMENT sequence. The following example illustrates this:

```
CREATE TABLE t1( c1 INT PRIMARY KEY, c2 INT DEFAULT AUTOINCREMENT );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 20 );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 20 );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 30 );
```

The row defined in the first INSERT statement is inserted, and c2 is set to 1. The row defined in the second INSERT statement is skipped because it matches the existing row. However, the autoincrement counter still increments to 2 (but does not impact the existing row). The row defined in the third INSERT statement is inserted, and the value of c2 is set to 3. So, the values inserted for the example above are:

```
20,1
30,3
```

> **⚠ Caution**
>
> If you are using SQL Remote, do not replicate DEFAULT LAST USER columns. When the column is replicated the column value is set to the SQL Remote user, not the replicated value.

OPTION clause

Use this clause to specify hints for executing the statement. The setting you specify is only applicable to the current statement and takes precedence over any public or temporary option settings, including those set by ODBC-enabled applications. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- `option-name` = `option-value`.
  Use an `OPTION( isolation_level = ... )` specification in the query text to override all other means of specifying isolation level for a query.
  Use an `OPTION( parameterization_level = ... )` specification in the query text to override the parameterization level for a query.

Remarks

The INSERT statement is used to add new rows to a database table.

Since text indexes and materialized views are impacted by changes to the underlying table data, consider truncating dependent text indexes or materialized views before bulk loading (LOAD TABLE, INSERT, MERGE) data into their underlying tables.

Insert one or more rows

Insert a single row, or multiple rows, with the specified expression column values. Multiple rows, if specified, are delimited by additional parentheses. The keyword DEFAULT can be used to cause the default value for the column to be inserted. If the optional list of column names is given, values are inserted one for one into the specified columns. If the list of column names is not specified, the values are inserted into the table columns in the order they were created (the same order as retrieved with SELECT *). The row is inserted into the table at an arbitrary position. (In relational databases, tables are not ordered.)

Insert the results of a SELECT statement

Perform mass insertion into a table with the results of a fully general SELECT statement. Insertions are done in an arbitrary order unless the SELECT statement contains an ORDER BY clause.

If you specify column names, the columns from the SELECT list are matched ordinally with the columns specified in the column list, or sequentially in the order in which the columns were created.

Inserts can be done into views if the query specification defining the view is updatable.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive. So, the string 'Value' inserted into a table is always held in the database with an uppercase V and the remainder of the letters lowercase. SELECT statements return the string as Value. If the database is not case sensitive, however, all comparisons make Value the same as value, VALUE, and so on. Further, if a single-column primary key already contains an entry Value, an INSERT of value is rejected, as it would make the primary key not unique.

Inserting a significant amount of data using the INSERT statement will also update column statistics.

> **i** Note
>
> To insert many rows into a table, it is more efficient to declare a cursor and insert the rows through the cursor, where possible, than to perform many separate INSERT statements. Before inserting data, you can specify the percentage of each table page that should be left free for later updates.

Privileges

You must be the owner of the table, or have the INSERT ANY TABLE privilege, or have INSERT privilege on the table. Additionally, if the ON EXISTING UPDATE clause is specified, you must have the UPDATE ANY TABLE system privilege, or have UPDATE privilege on the table.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Core Feature. The DEFAULT VALUES clause is optional ANSI/ISO SQL Language Feature F222, "INSERT statement: DEFAULT VALUES clause". Support for row value constructors in an INSERT statement comprises part of optional ANSI/ISO SQL Language Feature F641, "Row and table constructors". The VALUES keyword required by the software to specify the values to be inserted is not part of the standard.

The following extensions are also not part of the ANSI/ISO SQL Standard but are supported in the software:

- The INSERT...ON EXISTING clause is not in the standard. An ANSI/ISO SQL compliant equivalent in many instances is the MERGE statement.
- The OPTION clause.
- The WITH AUTO NAME clause.

Example

Add an Eastern Sales department to the database.

```
INSERT INTO GROUPO.Departments ( DepartmentID, DepartmentName )
VALUES ( 230, 'Eastern Sales' );
```

Create the table DepartmentHead and fill it with the names of department heads and their departments using the WITH AUTO NAME syntax.

```
CREATE TABLE DepartmentHead(
      pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
      DepartmentName VARCHAR(128),
      ManagerName VARCHAR(128) );
INSERT
INTO DepartmentHead WITH AUTO NAME
SELECT GivenName || ' ' || Surname AS ManagerName,
      DepartmentName
FROM GROUPO.Employees JOIN GROUPO.Departments
ON EmployeeID = DepartmentHeadID;
```

Create the table MyTable6 and populate it using the WITH AUTO NAME syntax.

```
CREATE TABLE MyTable6(
      pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
      TableName CHAR(128),
      TableNameLen INT );
INSERT INTO MyTable6 WITH AUTO NAME
SELECT
      length(t.table_name) AS TableNameLen,
```

```
      t.table_name AS TableName
FROM SYS.SYSTAB t
WHERE table_id <= 10;
```

Insert a new department, executing the statement at isolation level 3, rather than using the current isolation level setting of the database.

```
INSERT INTO GROUPO.Departments
   (DepartmentID, DepartmentName, DepartmentHeadID)
   VALUES(600, 'Foreign Sales', 129)
   OPTION( isolation_level = 3 );
```

The following example inserts three rows into a fictitious table, T:

```
INSERT INTO T (c1,c2,c3)
VALUES (1,10,100), (2,20,200), (3,30,300);
```

This example inserts three rows into a fictitious table, T, of four columns where each column has a default value defined:

```
INSERT INTO T ()
VALUES (), (), ();
```

Related Information

SQL Variables [page 123]
Data Import
Regular Views
Access to Data on Client Computers
Addition of Data Using INSERT
ALTER TABLE Statement [page 742]
MERGE Statement [page 1271]
INPUT Statement [Interactive SQL] [page 1224]
LOAD TABLE Statement [page 1247]
UPDATE Statement [page 1463]
SELECT Statement [page 1362]
TRUNCATE Statement [page 1442]
TRUNCATE TEXT INDEX Statement [page 1445]
DELETE Statement [page 1070]
PUT Statement [ESQL] [page 1313]

1.4.4.195 INSTALL EXTERNAL OBJECT Statement

Installs an object that can be run in an external environment.

Syntax

```
INSTALL EXTERNAL OBJECT object-name
[ update-mode ]
FROM { FILE file-path | VALUE expression }
ENVIRONMENT environment-name
[ AS USER user-name ]
```

```
environment-name :
PERL
| PHP
| JS
```

```
update-mode :
NEW
| UPDATE
```

Parameters

object-name

The name by which the installed object will be identified within the database.

update-mode

The update mode for the object. If the update mode is omitted, then NEW is assumed.

file-path

The location on the server computer from where the object is being installed.

environment-name

The name of the external environment in which the external object is run.

AS USER clause

Specifies the owner of the object.

Remarks

None.

Privileges

You must have the MANAGE ANY EXTERNAL OBJECT system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

In this example, you install a Perl script that is located in a file into the database.

```
INSTALL EXTERNAL OBJECT 'PerlScript'
NEW
FROM FILE 'perlfile.pl'
ENVIRONMENT PERL;
```

Perl code also can be built and installed from an expression, as follows:

```
INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE 'sub WriteToServerConsole { print $sa_output_handle $_[0]; }'
ENVIRONMENT PERL;
```

Perl code also can be built and installed from a variable, as follows:

```
CREATE VARIABLE PerlVariable LONG VARCHAR;
SET PerlVariable =
  'sub WriteToServerConsole { print $sa_output_handle $_[0]; }';
INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE PerlVariable
ENVIRONMENT PERL;
```

Related Information

External Environment Support
ALTER EXTERNAL ENVIRONMENT Statement [page 680]
REMOVE EXTERNAL OBJECT Statement [page 1333]
START EXTERNAL ENVIRONMENT Statement [page 1410]
STOP EXTERNAL ENVIRONMENT Statement [page 1423]
SYSEXTERNENV System View [page 1908]
SYSEXTERNENVOBJECT System View [page 1910]

1.4.4.196 INSTALL JAVA Statement

Makes Java classes available for use within a database.

Syntax

```
INSTALL JAVA
[ NEW | UPDATE ]
[ JAR jar-name ]
FROM { FILE filename | expression }
[ AS USER user-name ]
```

Parameters

NEW and UPDATE keyword clauses

If you specify NEW, the referenced Java class or JAR file must contain new classes, rather than updates of currently installed classes or JAR files. An error occurs if a class or JAR file with the same name exists in the database and NEW is used.

If you specify UPDATE, the referenced file may include replacements for Java classes or JAR files that are already installed in the given database.

If omitted, the default is NEW.

JAR clause

If this is specified, then the `filename` must designate a JAR file. JAR files typically have extensions of `.jar` or `.zip`.

Installed JAR and ZIP files can be compressed or uncompressed. Due to differences in compression schemes, it is recommended that JAR files containing textual resources be created with compression turned off.

If the JAR clause is specified, the JAR file is retained as a JAR after the classes that it contains have been installed. That JAR is the associated JAR of each of those classes. The JAR files installed in a database with the JAR clause are called the retained JAR files of the database.

The `jar-name` is a character string value, of up to 255 bytes long. The `jar-name` is used to identify the retained JAR in subsequent INSTALL JAVA UPDATE and REMOVE JAVA statements.

FROM FILE clause

Specifies the location of the Java class or JAR file to be installed.

The formats supported for `filename` include fully qualified file names, such as `'c:\\libs\\jarname.jar'` and `'/usr/u/libs/jarname.jar'`, and relative file names, which are relative to the current working directory of the database server. If the database server's class path includes the path to the class or JAR file, then the path does not need to be included with the file name. The database server's class path is defined by the -cp database server option and the java_class_path database option.

The `filename` must identify either a class, JAR or ZIP file.

FROM clause

Expressions must evaluate to a binary type whose value contains a valid class or JAR file.

AS USER clause

Specifies the owner of the object.

Remarks

The class definition for each class is loaded by each connection's VM the first time that class is used. When you INSTALL a class, the VM on your connection is implicitly restarted. Therefore, you have immediate access to the new class. Because the VM is restarted, any values stored in Java static variables are lost, and any SQL variables with Java class types are dropped.

For other connections, the new class is loaded the next time a VM accesses the class for the first time. If the class is already loaded by a VM, that connection does not see the new class until the VM is restarted for that connection.

All installed classes can be referenced in any way by any user.

Privileges

You must have the MANAGE ANY EXTERNAL OBJECT system privilege.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example installs a fictitious Java class named Demo, by providing the file name and location of the class.

```
INSTALL JAVA NEW
FROM FILE 'D:\\JavaClass\\Demo.class';
```

This example installs all the classes contained in a fictitious ZIP file, and associates them within the database with the JAR name Widgets. After the installation, the location of the ZIP file is not retained and classes must be referenced using the fully qualified class name (package name and class name).

```
INSTALL JAVA
JAR 'Widgets'
FROM FILE 'C:\\Jars\\Widget.zip';
```

Related Information

Installing a JAR File
Installing a Class File
REMOVE JAVA Statement [page 1334]
SYSJAR System View [page 1918]
SYSJARCOMPONENT System View [page 1919]
SYSJAVACLASS System View [page 1920]

1.4.4.197 INTERSECT Statement

Computes the intersection between the result sets of two or more queries.

Syntax

```
[ WITH temporary-views ] query-block
  INTERSECT [ ALL | DISTINCT ] query-block
[ ORDER BY [ integer | select-list-expression-name ] [ ASC | DESC ], ... ]
[ FOR XML xml-mode ]
[ OPTION( query-hint, ... ) ]
```

```
query-block : a query block
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| option-name = option-value
```

```
option-name : identifier
```

```
option-value :
hostvar (indicator allowed)
| string
| identifier
| number
```

Parameters

query-block

Query blocks are described in the documentation on common elements in SQL syntax.

FOR XML clause

The FOR XML clause is documented in the SELECT statement.

OPTION clause

Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- `option-name` = `option-value`.
  Use an `OPTION( isolation_level = ... )` specification in the query text to override all other means of specifying isolation level for a query.

Remarks

INTERSECT computes the set intersection between the result sets of two query blocks. Query blocks can be nested, and can in turn be comprised of nested SELECT statements or the set operators UNION, EXCEPT, or INTERSECT. Specifying INTERSECT alone is equivalent to specifying INTERSECT DISTINCT.

INTERSECT ALL implements bag intersection rather than set intersection. For example, if the first `query-block` contains 5 (duplicate) rows with specific values, and the second `query-block` contains 3 duplicate rows with identical values to the first, then INTERSECT ALL will return 3 rows.

The results of INTERSECT are the same as INTERSECT ALL if either `query-block` does not contain duplicate rows.

The two `query-block` result sets must be UNION-compatible; they must each have the same number of items in their respective SELECT lists, and the types of each expression should be comparable. If corresponding items in two SELECT lists have different data types, the database server chooses a data type for the corresponding column in the result and automatically convert the columns in each `query-block` appropriately.

The column names displayed are the same column names that are displayed for the first `query-block` and these names are used to determine the expression names to be matched with the ORDER BY clause. An alternative way of customizing result set column names is to use a common table expression (the WITH clause).

Privileges

You must have the SELECT ANY TABLE system privilege, or be the owner of the objects specified in `query-block`, or have SELECT privileges on each query block.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

INTERSECT is optional ANSI/ISO SQL Language Feature F302. Explicitly specifying the DISTINCT keyword with INTERSECT is optional ANSI/ISO SQL Language Feature T551. Specifying an ORDER BY clause with INTERSECT is SQL Language Feature F850. A `query-block` that contains an ORDER BY clause constitutes ANSI/ISO SQL Feature F851. A query block that contains a row-limit clause (SELECT TOP or LIMIT) comprises optional ANSI/ISO SQL Language Feature F857 or F858, depending on the context. The FOR XML and OPTION clauses are not part of the Standard.

Transact-SQL

INTERSECT is not supported by Adaptive Server Enterprise. However, both INTERSECT ALL and INTERSECT DISTINCT can be used in the Transact-SQL dialect supported by SQL Anywhere.

Related Information

Set Operators and the NULL Value
EXCEPT Statement [page 1148]
UNION Statement [page 1450]
SELECT Statement [page 1362]

1.4.4.198 LEAVE Statement

Leaves a compound statement or loop.

Syntax

```
LEAVE [ statement-label ]
```

Remarks

The LEAVE statement is a control statement that leaves a labeled (`statement-label`) compound statement or a labeled loop. LEAVE can take a label that is either the beginning of a loop or a BEGIN...END block. If no `statement-label` is not specified, then LEAVE exits the innermost loop not the innermost BEGIN...END block. Execution resumes at the first statement after the compound statement or loop.

Using the LEAVE statement without a label is equivalent to using the Transact-SQL BREAK statement.

A compound statement that is the body of a procedure or trigger has an implicit label that is the same as the name of the procedure or trigger.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Part of optional Language Feature P002, "Computational completeness".

Example

The following fragment shows how the LEAVE statement is used without a label to leave a loop.

```
SET i = 1;
lbl:
LOOP
   INSERT
   INTO Counters ( number )
   VALUES ( i );
   IF i >= 10 THEN
      LEAVE;
   END IF;
   SET i = i + 1
END LOOP lbl
```

The following fragment uses LEAVE in a nested loop.

```
outer_loop:
LOOP
   SET i = 1;
   inner_loop:
   LOOP
      ...
      SET i = i + 1;
      IF i >= 10 THEN
         LEAVE outer_loop
      END IF
   END LOOP inner_loop
END LOOP outer_loop
```

Related Information

Stored Procedures, Triggers, Batches, and User-defined Functions
LOOP Statement [page 1269]
FOR Statement [page 1166]
BEGIN Statement [page 784]

1.4.4.199 LOAD STATISTICS Statement

Intended for internal use, this statement is used by the dbunload utility to unload column statistics from the old database into the ISYSCOLSTAT system table.

Syntax

```
LOAD STATISTICS [ [ owner.]table-name.]column-name
 format-id, density, max-steps, actual-steps, step-values, frequencies
```

Parameters

format-id

Internal field used to determine the format of the rest of the row in the ISYSCOLSTAT system table.

density

An estimate of the weighted average selectivity of a single value for the column, not counting the selectivity of large single value selectivities stored in the row.

max-steps

The maximum number of steps allowed in the histogram.

actual-steps

The number of steps actually used at this time.

step-values

Boundary values of the histogram steps.

frequencies

Selectivities of histogram steps.

Privileges

You must have the MANAGE ANY STATISTICS system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

SYSCOLSTAT System View [page 1902]
Unload Utility (dbunload)

1.4.4.200 LOAD TABLE Statement

Imports bulk data into a database table from an external file.

Syntax

```
LOAD [ INTO ] TABLE [ owner.]table-name |view-name
[ ( column-name, ... ) ]
load-source
[ load-option ... ]
[ statistics-limitation-option ]
```

```
load-source :
{ FROM filename-expression
   | USING FILE filename-expression
   | USING CLIENT FILE client-filename-expression
   | USING VALUE value-expression
   | USING COLUMN column-expression }
```

```
filename-expression : string | variable
```

```
client-filename-expression : string | variable
```

```
value-expression : expression
```

```
column-expression :
column-name
   FROM table-name
   ORDER BY column-list
```

```
load-option :
ALLOW { integer | ALL | NO } ERRORS ]
| BYTE ORDER MARK { ON | OFF }
| CHECK CONSTRAINTS { ON | OFF }
| { COMPRESSED | AUTO COMPRESSED | NOT COMPRESSED }
| COMMENTS INTRODUCED BY comment-prefix
| COMPUTES { ON | OFF }
| DEFAULTS { ON | OFF }
| DELIMITED BY string
| ENCODING encoding
| { ENCRYPTED KEY 'key' | NOT ENCRYPTED }
| ESCAPE CHARACTER character
| ESCAPES { ON | OFF }
| FORMAT {
    TEXT
    | BCP
    | XML row-xpath ( column-xpath,...) [ NAMESPACES namespace ] }
    | SHAPEFILE
| HEXADECIMAL { ON | OFF }
| MESSAGE LOG log-target
| ORDER { ON | OFF }
| PCTFREE percent-free-space
| QUOTE string
| QUOTES { ON | OFF }
| ROW DELIMITED BY string
| ROW LOG log-target
| SKIP integer
| STRIP { OFF | LTRIM | RTRIM | BOTH }
| WITH CHECKPOINT { ON | OFF }
| WITH { FILE NAME | ROW | CONTENT } LOGGING
```

```
statistics-limitation-option :
STATISTICS {
    ON [ ALL COLUMNS ]
    | ON KEY COLUMNS
    | ON ( column-list )
    | OFF
    }
```

```
comment-prefix : string
```

```
encoding : string
```

```
log-target : {
FILE server-filename
| CLIENT FILE client-filename
| VARIABLE variable-name
}
```

Parameters

view-name

Use this clause to specify a regular view to load data into. The view definition must be based on a base table. The view definition must be defined as follows:

```
CREATE VIEW
[owner.]view-name
AS SELECT * FROM base-table-name
```

column-name

Use this clause to specify one or more columns to load data into. Any columns not present in the column list become NULL if DEFAULTS is OFF. If DEFAULTS is ON and the column has a default value, then that value is used. If DEFAULTS is OFF and a non-nullable column is omitted from the column list, then the database server attempts to convert the empty string to the column's type.

When a column list is specified, it lists the columns that are expected to exist in the file and the order in which they are expected to appear. Column names cannot be repeated. Columns whose names do not appear in the list are set to NULL/zero/empty or DEFAULT (depending on column nullability, data type, and the DEFAULTS setting). Columns that exist in the input file that are to be ignored by LOAD TABLE can be specified using **`filler()`** as a column name.

load-source

Use this clause to specify the data source to load data from. There are several sources of data from which data can be loaded.

When the LOAD TABLE statement is inside the BEGIN PARALLEL WORK statement, then the `load-source` parameter only supports the USING FILE clause and the FROM clause.

FROM clause

Use this to specify a file. The `filename-expression` is passed to the database server as a string. The string is therefore subject to the same database formatting requirements as other SQL strings. In particular:

- To indicate directory paths, the backslash character (\) must be represented by two backslashes.
- The path name is relative to the database server, not to the client application.
- You can use UNC path names to load data from files on computers other than the database server.

USING FILE clause

Use this clause to load data from a file. This is synonymous with specifying the FROM `filename` clause.

When the LOAD TABLE statement is used with the USING FILE clause, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

USING CLIENT FILE clause

Use this clause to load data from a file on a client computer. When the database server retrieves data from `client-filename-expression`, the data is not materialized in the server's memory, so the database server limit on the size of BLOB expressions does not apply to the file. Therefore, the client file can be of an arbitrary size.

Client-side data transfers are supported for clients that use the SQL Anywhere database drivers. They are not supported by the **Tabular Data Stream** (**TDS**) protocol used by SAP Adaptive Server Enterprise, the jConnect JDBC driver, or SAP Open Client applications.

File name logging is not allowed if the table is being loaded from a client file. If the logging type is not specified, then WITH CONTENT LOGGING is used.

When the LOAD TABLE statement is used with the USING CLIENT FILE clause, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

Use the ClientFileValidator connection parameter to control when the database server is permitted to read and write local files.

USING VALUE clause

Use this clause to load data from any expression of CHAR, NCHAR, BINARY, or LONG BINARY type, or BLOB string. The following are examples of how this clause can be used:

- The following syntax uses the xp_read_file system procedure to get the values to load from the target file:

  ```
  ... USING VALUE xp_read_file( 'filename' )...
  ```

- The following syntax specifies the value directly, inserting two rows with values of 4 and 5, respectively;

  ```
  ... USING VALUE '4\n5'...
  ```

- The following syntax uses the results of the READ_CLIENT_FILE function as the value:

  ```
  ... USING VALUE READ_CLIENT_FILE( client-filename-expression )
  ```

  In this case, you can also specify `USING CLIENT FILE client-filename-expression` since they are semantically equivalent.

If the ENCODING clause is not specified in the LOAD TABLE statement, then encoding for `value-expression` is assumed to be in the database character set (db_charset) if `value-expression` is of type CHAR or BINARY, and NCHAR database character set (nchar_charset) if `value-expression` is of type NCHAR.

USING COLUMN clause

Use this clause to load data from a single column in another table. This clause is used by the database server when it replays the transaction log during recovery by replaying the LOAD TABLE...WITH CONTENT LOGGING statements. Transaction log records for LOAD TABLE...WITH CONTENT LOGGING statements comprise chunks of the original input file. When the database server encounters these chunks in the transaction log during recovery, it loads the chunks into a temporary table and then loads all the data from the original load operation.

The following items are required in the USING COLUMN clause:

table-name

The name of the base or temporary table that contains the column to load data from. When used by the database server during recovery from the transaction log, this is the table that holds the chunks of rows to be parsed and loaded.

column-name

The name of the column in `table-name` that holds the chunks of rows to be loaded.

column-list

One or more columns in the destination table used to sort the rows before loading the data. `column-list` must be a verifiable unique set of values, such as a primary key or a unique index on non-nullable columns included within the column list.

load-option clause

There are several load options you can specify to control how data is loaded. The following list gives the supported load options:

ALLOW (integer | ALL | NO) ERRORS clause

This clause can only be specified once for the statement. The default value for this clause is 0, which means that a violation generates an error and the statement is rolled back. If you specify an integer, `n`, then on error `n`+1 the database server rolls back the statement. The values ALLOW NO ERRORS and ALLOW 0 ERRORS are equivalent. This clause allows the database server to set problematic data aside and progress with the load operation.

The database server reports the last error that was encountered to the user, and this error is also logged to the MESSAGE log. Rows that are written to the ROW log can be changed and used as input to a subsequent LOAD TABLE statement.

If a ROW LOG is written to a database server or client file, then its contents are written in the same character set as the original input file. If a MESSAGE LOG is written to a server or client file, then its contents are written in the client's language and in the client connection's CHAR character set. If a ROW or MESSAGE LOG is written to a CHAR or NCHAR variable, then it is written in the CHAR or NCHAR (respectively) character set.

BYTE ORDER MARK clause

Use this clause to specify whether the server should search for and interpret a byte order mark (BOM) at the beginning of the data. By default, this option is ON. If BYTE ORDER MARK is OFF, then the server does not search for a BOM.

If the ENCODING clause is specified:

- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, then the database server searches for a BOM at the beginning of the data. If a BOM is present, then it is used to verify the endianness of the data. If you specify the wrong endian, then an error is returned.
- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, then the database server searches for a BOM at the beginning of the data. If a BOM is present, then it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.
- If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, then the database server searches for a BOM at the beginning of the data. If a BOM is present, then it is ignored.

If the ENCODING clause is not specified:

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, then the server looks for a BOM at the beginning of the input data. If a BOM is located, then the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or UTF-8) and the BOM is not considered to be part of the data to be loaded. If no BOM is found, the database CHAR encoding is used.

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, then the database CHAR encoding is used, and the database server does not look for or interpret a BOM at the beginning of the data.

CHECK CONSTRAINTS clause

Use this clause to control whether constraints are checked during loading. CHECK CONSTRAINTS is ON by default, but the Unload utility (dbunload) writes out LOAD TABLE statements with CHECK CONSTRAINTS set to OFF. Setting CHECK CONSTRAINTS to OFF disables check constraints, which can be useful, for example, during database rebuilding. If a table has check constraints that call user-defined functions that are not yet created, then the rebuild fails unless CHECK CONSTRAINTS is set to OFF.

COMMENTS INTRODUCED BY clause

Use this clause to specify the string used in the data file to introduce a comment. When used, LOAD TABLE ignores any line that begins with the string `comment-prefix`.

Comments are only allowed at the beginning of a new line.

If COMMENTS INTRODUCED BY is omitted, then the data file must not contain comments because they are interpreted as data.

COMPRESSED clause

Specify COMPRESSED if the data being loaded is compressed in the input file. The database server decompresses the data before loading it. If you specify COMPRESSED and the data is not compressed, then the LOAD fails and returns an error.

Specify AUTO COMPRESSED to allow the database server determine whether the data in the input file is compressed. If so, the database server decompresses the data before loading it.

Specify NOT COMPRESSED to indicate that the data in the input file is not compressed. You can also specify NOT COMPRESSED if the data is compressed, but you don't want the database server to decompress it. In this case, the data remains compressed in the database. However, if a file is both encrypted and compressed, then you cannot use NOT ENCRYPTED without also using NOT COMPRESSED.

COMPUTES clause

By default, this option is ON, which enables recalculation of computed columns. Setting COMPUTES to OFF disables computed column recalculations. COMPUTES OFF is useful, for example, if you are rebuilding a database, and a table has a computed column that calls a user-defined function that is not yet created. The rebuild would fail unless this option was set to OFF.

The Unload utility (dbunload) writes out LOAD TABLE statements with the COMPUTES set to OFF.

DEFAULTS clause

By default, DEFAULTS is set to OFF. If DEFAULTS is OFF, then any column not present in the list of columns is assigned NULL. If DEFAULTS is set to OFF and a non-nullable column is omitted from the list, then the database server attempts to convert the empty string to the column's type. If DEFAULTS is set to ON and the column has a default value, then that value is used.

DELIMITED BY clause

Use this clause to specify the column delimiter string. The default column delimiter string is a comma; however, it can be any string up to 255 bytes in length (for example, `... DELIMITED BY '###' ...`). The delimiter you specify is a string and should be quoted. To specify tab-delimited

values, you could specify the hexadecimal escape sequence for the tab character (9), `... DELIMITED BY '\x09' ...`.

ENCODING clause

Use this clause to specify the character encoding used for the data being loaded into the database. The ENCODING clause cannot be used with the BCP format.

If a translation error occurs during the load operation, then it is reported based on the setting of the on_charset_conversion_failure option.

Specify the BYTE ORDER clause to interpret a byte order mark in the data.

If the ENCODING clause is specified:

- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, then the database server searches for a BOM at the beginning of the data. If a BOM is present, then it is used to verify the endianness of the data. If you specify the wrong endian, then an error is returned.
- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, then the database server searches for a BOM at the beginning of the data. If a BOM is present, then it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.
- If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, then the database server searches for a BOM at the beginning of the data. If a BOM is present it is ignored.

If the ENCODING clause is not specified:

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, then the server looks for a BOM at the beginning of the input data. If a BOM is located, then the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or UTF-8) and the BOM is not considered to be part of the data to be loaded.
- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, then the database CHAR encoding is used, and the database server does not look for or interpret a BOM at the beginning of the data.

ENCRYPTED clause

Use this clause to specify encryption settings. When loading encrypted data, specify ENCRYPTED KEY followed by the key used to encrypt the data in the input file. The key can be either a string or a variable name.

Specify NOT ENCRYPTED to indicate that the data in the input file is not encrypted. You can also specify NOT ENCRYPTED if the data is encrypted, but you don't want the database server to decrypt it. In this case, the data remains encrypted in the database. However, if a file is both encrypted and compressed, then you cannot use NOT ENCRYPTED without also using NOT COMPRESSED.

ESCAPE CHARACTER clause

Use this clause to specify the escape character used in the data. The default escape character for characters stored as hexadecimal codes and symbols is a backslash (\), so \x0A is the line feed character, for example. This can be changed using the ESCAPE CHARACTER clause. For example, to use the exclamation mark as the escape character, you would enter:

```
ESCAPE CHARACTER '!'
```

It is recommended that the string you specify for the escape character is no longer than one multibyte character.

ESCAPES clause

Use this clause to control whether to recognize escape characters. With ESCAPES turned ON (the default), characters following the escape character (which defaults to \) are recognized and interpreted as special characters by the database server. Newline characters can be included as the combination \n, and other characters can be included in data as hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\ ) is interpreted as a single backslash. A backslash followed by any character other than n, x, X, or \ is interpreted as two separate characters. For example, \q inserts a backslash and the letter q. It is recommended that the string you specify for the escape character is no longer than one multibyte character.

FORMAT TEXT

If you choose FORMAT TEXT (the default), input lines are assumed to be characters (as defined by the ENCODING option), one row per line, with values separated by the column delimiter string.

FORMAT BCP

Specify FORMAT BCP to load Adaptive Server Enterprise-generated BCP out files.

FORMAT SHAPEFILE

Specify FORMAT SHAPEFILE to load ESRI shapefiles. The shapefile must be on the database server computer and must be loaded using FROM `filename-expression` or USING FILE `filename-expression`, where `filename-expression` refers to an ESRI shapefile with the `.shp` file extension. The associated `.shx` and `.dbf` files must be located in the same directory as the `.shp` file, and have the same base file name.

For FORMAT SHAPEFILE, the encoding defaults to ISO-8859-1 if the ENCODING clause is not specified.

If you specify FORMAT SHAPEFILE, then the following load options are allowed:

- CHECK CONSTRAINTS
- COMPUTES
- DEFAULTS
- ENCODING
- ORDER
- PCTFREE
- WITH CHECKPOINT
- WITH .... LOGGING

The LOAD TABLE statement gets the SRID from the second column type that you are loading into. For example, if you created the second column with type ST_Geometry(SRID=4326), then the geometries are loaded using SRID 4326. If your second column has type ST_Geometry (no explicit SRID), then the geometries are loaded using SRID 0.

FORMAT XML

If you specify FORMAT XML, then the following load options are allowed:

- BYTE ORDER MARK
- CHECK CONSTRAINTS
- COMPRESSED

- COMPUTES
- DEFAULTS
- ENCODING
- ENCRYPTED
- ORDER
- PCTFREE
- WITH CHECKPOINT
- WITH...LOGGING

If you use FORMAT XML, then the input file is parsed in the same way as a query that uses the OPENXML operator. The arguments of the SQL statement correspond to the system procedure parameters as follows:

LOAD TABLE statement clause	OPENXML operator argument	Details
`row-xpath`	`xpath`	
-	`flags`	There is no way to specify a value with FORMAT XML that corresponds to the `flags` argument of OPENXML.
NAMESPACES	`namespaces`	

The FORMAT XML clause uses the following parameters:

row-xpath

A string or variable containing an XPath query. XPath allows you to specify patterns that describe the structure of the XML document you are querying. The XPath pattern included in this argument selects the nodes from the XML document. Each node that matches the XPath query in the `row-xpath` argument generates one row in the table.

Metaproperties can only be specified in FORMAT XML clause `row-xpath` arguments. A metaproperty is accessed within an XPath query as if it was an attribute. If `namespaces` is not specified, then by default the prefix mp is bound to the Uniform Resource Identifier (URI) urn:sap-com:sa-xpath-metaprop. If `namespace` is specified, then this URI must be bound to mp or some other prefix to access metaproperties in the query. Metaproperty names are case sensitive. The following metaproperties are supported:

@mp:id

returns an ID for a node that is unique within the XML document. The ID for a given node in a given document may change if the database server is restarted. The value of this metaproperty increases with document order.

@mp:localname

returns the local part of the node name, or NULL if the node does not have a name.

@mp:prefix

returns the prefix part of the node name, or NULL if the node does not have a name or if the name is not prefixed.

@mp:namespaceuri

returns the URI of the namespace that the node belongs to, or NULL if the node is not in a namespace.

@mp:xmltext

returns a subtree of the XML document in XML form. For example, when you match an internal node, you can use this metaproperty to return an XML string, rather than the concatenated values of the descendant text nodes.

column-xpath

A string or variable that specifies the schema of the result set and how the value is found for each column in the result set. If a FORMAT XML clause expression matches more than one node, then only the first node in the document order is used. If the node is not a text node, then the result is found by appending all the text node descendants. If a FORMAT XML clause expression does not match any nodes, then the column for that row is NULL.

namespace

A string or variable containing an XML document. The in-scope namespaces for the query are taken from the root element of the document.

HEXADECIMAL clause

Use this clause to specify whether to read binary values as hexadecimal values. By default, HEXADECIMAL is ON. With HEXADECIMAL ON, binary column values are read as **0x** `nnnnnn`..., where 0x is a zero followed by an x, and each `n` is a hexadecimal digit. It is important to use HEXADECIMAL ON when dealing with multibyte character sets.

The HEXADECIMAL clause can be used only with the FORMAT TEXT clause.

MESSAGE LOG clause

This clause can only be specified once for the statement. When an error is encountered while inserting or parsing a row, the database server writes the error to the specified location.

ORDER clause

Use this clause to specify whether to sort the data when loading. The default for ORDER is ON. If ORDER is ON, and a clustered index has been declared, then LOAD TABLE sorts the input data according to the clustered index and inserts rows in the same order. If the data you are loading is already sorted, then set ORDER to OFF.

PCTFREE clause

Use this clause to specify the percentage of free space you want to reserve for each table page. This setting overrides any permanent setting for the table, but only for the duration of the load, and only for the data being loaded. The value `percent-free-space` is an integer between 0 and 100. A value of 0 specifies that no free space is to be left on each page. Each page is to be fully packed. A high value causes each row to be inserted into a page by itself.

QUOTE clause

The QUOTE clause is for TEXT data only; the `string` is placed around string values. The default is a single quote (apostrophe).

QUOTES clause

Use this clause to specify whether strings are enclosed in quotes. When QUOTES is set to ON (the default), the LOAD TABLE statement expects strings to be enclosed in quote characters. If the QUOTES clause is omitted, then the quote character is either an apostrophe (single quote) or a quotation mark (double quote) and the first such character encountered in a string is treated as the quote character for the string. Strings must be terminated by a matching quote.

When QUOTES is set to ON, column delimiter strings can be included in column values. Also, quote characters are assumed not to be part of the value. Therefore, the following line is treated as two values, not three, despite the presence of the comma in the address. Also, the quotes surrounding the address are not inserted into the database.

```
'123 High Street, Anytown',(715)398-2354
```

To include a quote character in a value, when QUOTES is set to ON, you must use two quotes. The following line includes a value in the third column that is a single quote character:

```
'123 High Street, Anytown','(715)398-2354',''''
```

ROW DELIMITED BY clause

Use this clause to specify the string that indicates the end of an input record. The default delimiter string is a newline (\n); however, it can be any string up to 255 bytes in length (for example, `ROW DELIMITED BY '###'`). If you wanted to specify tab-delimited values, then you could specify the hexadecimal escape sequence for the tab character (9), `ROW DELIMITED BY '\x09'`. If your delimiter string contains a \n, then it matches either \r\n or \n.

ROW LOG clause

This clause can only be specified once for the statement. When an error is encountered while inserting or parsing a row, the database server writes an image of the input row to the specified location in addition to reporting the row to the user.

SKIP clause

Use this clause to specify whether to ignore lines at the beginning of a file. The `integer` argument specifies the number of lines to skip. You can use this clause to skip over a line containing column headings, for example. If the row delimiter is not the default (newline), then skipping may not work correctly if the data contains the row delimiter embedded within a quoted string.

STRIP clause

Use this clause to specify whether unquoted values should have leading or trailing blanks stripped off before they are inserted. The STRIP option accepts the following options:

STRIP OFF

Do not strip off leading or trailing blanks.

STRIP LTRIM

Strip leading blanks.

STRIP RTRIM

Strip trailing blanks.

STRIP BOTH

Strip both leading and trailing blanks.

The STRIP behavior is tied to the QUOTES clause. If you specify QUOTES OFF, then STRIP OFF, STRIP LTRIM, STRIP RTRIM, and STRIP BOTH work exactly as their wording suggests. If you do not specify a QUOTES clause or you specify QUOTES ON, then unquoted strings are always left-trimmed and right-trimmed (however, you can specify STRIP OFF or STRIP LTRIM if you don't want the strings to be right-trimmed as well).

WITH CHECKPOINT clause

Use this clause to specify whether to perform a checkpoint. The default setting is OFF. If this clause is set to ON, then a checkpoint is issued after successfully completing and logging the statement. If this clause is set to ON and the database requires automatic recovery before a checkpoint is issued, then the data file used to load the table must be present for the recovery to complete successfully if you use FILE NAME LOGGING. If WITH CHECKPOINT ON is specified and recovery is subsequently required, then recovery begins after the checkpoint, and the data file does not need to be present.

The data files are required, regardless of what is specified for this clause, if the database becomes corrupt and you must use a backup and apply the current log file if you use FILE NAME LOGGING.

When the LOAD TABLE statement is inside the BEGIN PARALLEL WORK statement, the WITH CHECKPOINT ON clause is not supported. However a checkpoint executes at the beginning of a BEGIN WORK PARALLEL statement.

WITH { FILE NAME | ROW | CONTENT } LOGGING

Use this clause to control the level of detail logged in the transaction log during a load operation.

The levels of logging are as follows:

WITH FILE NAME LOGGING clause

Inserts are not recorded in the transaction log file unless WITH ROW LOGGING clause is specified, so the inserted rows may not be recovered in the event of a failure depending upon the logging type.

When you do not specify a logging level in the LOAD TABLE statement, WITH ROW LOGGING is the default level when specifying:

- FROM `filename-expression`
- USING FILE `filename-expression`

WITH ROW LOGGING clause

The WITH ROW LOGGING clause causes each row that is loaded to be recorded in the transaction log as an INSERT statement. This level of logging is recommended for databases involved in synchronization and is the default for database mirroring when using FROM `filename-expression` or USING FILE `filename-expression`. However, when loading large amounts of data, this logging type can affect performance, and results in a much longer transaction log.

If there are no non-deterministic values, WITH CONTENT LOGGING likely results in better performance

This level is also ideal for databases where the table being loaded into contains non-deterministic values, such as computed columns, or CURRENT TIMESTAMP defaults.

When the LOAD TABLE statement is inside the BEGIN PARALLEL WORK statement, this clause is not supported.

WITH CONTENT LOGGING clause

The WITH CONTENT LOGGING clause causes the database server to copy the input file to the transaction log in chunks. These chunks can be reconstituted into a copy of the input file later, for example during recovery from the transaction log. When loading large amounts of data, this logging type has a very low impact on performance, and offers increased data protection, but it does result in a longer transaction log. This level of logging is recommended for databases involved in mirroring, or where it is desirable to not maintain the original data files for later recovery provided there are no non-deterministic values.

The WITH CONTENT LOGGING clause cannot be used if the database is involved in synchronization. The WITH CONTENT LOGGING clause is required if the table is being loaded from a client file.

When you do not specify a logging level in the LOAD TABLE statement, WITH CONTENT LOGGING is the default level when specifying:

- USING CLIENT FILE `client-filename-expression`
- USING VALUE `value-expression`
- USING COLUMN `column-expression`

Similarly, when using a LOAD TABLE statement on a primary or root server, WITH CONTENT LOGGING is the default.

When the LOAD TABLE statement is inside the BEGIN PARALLEL WORK statement, this clause is not supported.

statistics-limitation-option

Allows you to limit the columns for which statistics are generated during the execution of LOAD TABLE. Otherwise, statistics are generated for all columns. Use this clause only if you are certain that statistics will not be used on some columns. You can specify ON ALL COLUMNS (the default), OFF, ON KEY COLUMNS, or a list of columns for which statistics should be generated.

Remarks

LOAD TABLE allows efficient mass insertion into a database table from a file. LOAD TABLE is more efficient than the Interactive SQL statement INPUT.

You can execute multiple concurrent LOAD TABLE statements when you run in never-write mode using the -im database server option.

You can concurrently execute a list of LOAD TABLE statements by using the BEGIN PARALLEL WORK statement. When the LOAD TABLE statement is inside a BEGIN PARALLEL WORK statement:

- You can only load content into base tables or global temporary tables that can be shared by all connections to the database.
- Each LOAD TABLE statement must specify a different target table.
- If the wait_for_commit database option is enabled for your connection, you must disable it before executing the BEGIN PARALLEL WORK statement.

LOAD TABLE uses an exclusive schema lock. For base tables, global temporary tables, and local temporary tables, a commit is performed.

If you attempt to use LOAD TABLE on a table on which an immediate text index is built, or that is referenced by an immediate view, then the load fails. This does not occur for non-immediate text indexes or materialized views; however, it is strongly recommended that you truncate the data in dependent indexes and materialized views before executing the LOAD TABLE statement, and then refresh the indexes and views after.

Do not use the LOAD TABLE statement on a temporary table for which ON COMMIT DELETE ROWS was specified, either explicitly or by default, at creation time. However, you *can* use LOAD TABLE if ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL was specified.

With FORMAT TEXT, a NULL value is indicated by specifying no value. For example, if three values are expected and the file contains `1,,'Fred',`, then the values inserted are 1, NULL, and Fred. If the file contains `1,2,`, then the values 1, 2, and NULL are inserted. Values that consist only of spaces are also considered NULL values. For example, if the file contains `1, ,'Fred',`, then values 1, NULL, and Fred are inserted. All other values are considered not NULL. For example, '' (a single quote followed by single quote) is an empty string. 'NULL' is a string containing four letters.

If a column being loaded by LOAD TABLE does not allow NULL values and the file value is NULL, then numeric columns are given the value 0 (zero), character columns are given an empty string (''). If a column being loaded by LOAD TABLE allows NULL values and the file value is NULL, then the column value is NULL (for all types).

If the table contains columns a, b, and c, and the input data contains a, b, and c, but the LOAD statement specifies only a and b as columns to load data into, then the following values are inserted into column c:

- if DEFAULTS ON is specified, and column c has a default value, the default value is used.
- if column c does not have a default defined for it and it allows NULLs, then a NULL is used.
- if column c does not have a default defined for it and it does not allow NULLs, then either a zero (0) or an empty string ('') is used, or an error is returned, depending on the data type of the column.

To create histograms on table columns, LOAD TABLE captures column statistics when it loads data. The histograms are used by the optimizer. Following are additional tips about loading and column statistics:

- LOAD TABLE saves statistics on base tables for future use. It does not save statistics on global temporary tables.
- If you are loading into an empty table that may have previously contained data, then it may be beneficial to drop the statistics for the column before executing the LOAD TABLE statement.
- If column statistics exist when LOAD TABLE is performed on a column, then statistics for the column are *not* recalculated. Instead, statistics for the new data are inserted into the existing statistics. If the existing column statistics are out-of-date, then they are still out of date after loading new data into the column. If you suspect that the column statistics are out of date, then consider updating them either before, or after, executing the LOAD TABLE statement.
- LOAD TABLE adds statistics only if the table has five or more rows. If the table has at least five rows, then histograms are modified as follows:

Data already in table?	Histogram present?	Action taken
Yes	Yes	Integrate changes into the existing histograms
Yes	No	Do not build histograms
No	Yes	Integrate changes into the existing histograms
No	No	Build new histograms

- LOAD TABLE does not generate statistics for columns that contain NULL values for more than 90% of the rows being loaded.

You can execute a LOAD TABLE statement with a dynamically constructed file name by assigning the file name to a variable and using the variable name in the LOAD TABLE statement.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

Privileges

The required privilege depend on the -gl server option.

If the -gl option is set to ALL, then one of the following must be true:

- you are the owner of the table
- you have LOAD privilege on the table
- you have the LOAD ANY TABLE system privilege
- you have the ALTER ANY TABLE system privilege

If the -gl option is set to DBA, then you must have the LOAD ANY TABLE or ALTER ANY TABLE system privilege.

If the -gl option is set to NONE, then LOAD TABLE is not permitted.

When loading into a view, you must have one of the following must be true:

- you have the LOAD ANY TABLE system privilege
- you have the LOAD privilege on the view, as well as the LOAD privilege on the base table, that the view is dependent on.

When loading from a file on a client computer:

- READ CLIENT FILE privilege is also required.
- Read privileges are required on the directory being read from.
- The allow_read_client_file database option must be enabled.
- The READ_CLIENT_FILE feature must be enabled.

Side Effects

Automatic commit.

The original file is required if you must recover the rows and WITH FILE NAME LOGGING is used. In addition, the LOAD TABLE statement should not be used without the WITH ROW LOGGING clause in databases that are used as MobiLink clients, or in a database involved in SQL Remote replication, because these technologies replicate changes through analysis of the transaction log file.

The LOAD TABLE statement does not fire any triggers associated with the table.

A checkpoint is carried out at the beginning of the operation. A second checkpoint is performed at the end if WITH CHECKPOINT ON is specified.

Column statistics are updated if a significant amount of data is loaded.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Suppose you create a table, myTable, as follows:

```
CREATE TABLE myTable( a CHAR(100), let_me_default INT DEFAULT 1, c CHAR(100) );
```

Then you create an input file called `c:\temp\input.txt` and put the following data in it:

```
ignore_me, this_is_for_column_c, this_is_for_column_a
```

Now, you load the data from `c:\temp\input.txt` into myTable as follows:

```
LOAD TABLE myTable ( filler(), c, a ) FROM 'c:\\temp\\input.txt' FORMAT TEXT
DEFAULTS ON;
```

The command `SELECT * FROM myTable` yields the result set:

a	let_me_default	c
this_is_for_column_a	1	this_is_for_column_c

The following example executes the LOAD TABLE statement with a dynamically constructed file name, via the EXECUTE IMMEDIATE statement:

```
CREATE PROCEDURE LoadData( IN from_file LONG VARCHAR )
BEGIN
    DECLARE path LONG VARCHAR;
    SET path = 'd:\\data\\' || from_file;
    LOAD TABLE MyTable FROM path;
END;
```

The following example loads UTF-8-encoded table data from a file into mytable:

```
LOAD TABLE mytable FROM 'c:\\temp\\mytable_data_in_utf8.dat' ENCODING 'UTF-8';
```

In this example, lines in the file `c:\temp\input2.dat` that start with *//* are ignored.

```
LOAD TABLE GROUPO.Employees FROM 'c:\\temp\\input2.dat' COMMENTS INTRODUCED BY
'//'
```

In the following example, a table is created and filled from the data stored in strings.

```
CREATE OR REPLACE TABLE tbltarget (col1 INTEGER PRIMARY KEY, col2 DATETIME);
```

```
LOAD INTO TABLE tbltarget
    USING VALUE '1,2018-09-13 11:22:33\n2,2018-09-13 11:23:33\n3,2018-09-13
11:24:34\n';
LOAD INTO TABLE tbltarget
    USING VALUE '4,2018-09-12 13:52:33\n5,2018-09-12 13:53:33\n6,2018-09-12
13:54:34\n';
SELECT * FROM tbltarget;
```

Column values (for example, 2,2018-09-13 11:23:33) in the strings are separated into individual rows by newline characters (\n).

In the following example, a table is filled with data in a special format so that the USING COLUMN clause can be demonstrated next.

```
CREATE OR REPLACE TABLE tbldata (keyval UNSIGNED INT PRIMARY KEY, dataval LONG
VARCHAR NOT NULL);
INSERT INTO tbldata (keyval, dataval)
    VALUES (2,'1,2018-09-13 11:22:33\n2,2018-09-13 11:23:33\n3,2018-09-13
11:24:34\n');
INSERT INTO tbldata (keyval, dataval)
    VALUES (1,'4,2018-09-12 13:52:33\n5,2018-09-12 13:53:33\n6,2018-09-12
13:54:34\n');
```

Column values (for example, 2,2018-09-13 11:23:33) in the data table are separated into individual rows by newline characters (\n).

The target table is filled from the data stored in the source table.

```
CREATE OR REPLACE TABLE tbltarget (col1 INTEGER PRIMARY KEY, col2 DATETIME);
LOAD INTO TABLE tbltarget
    USING COLUMN dataval FROM tbldata
    ORDER BY keyval ENCODING 'windows-1252';
SELECT * FROM tbltarget;
```

This syntax is commonly used in the transaction log to enable recovery of table data.

Related Information

Data Import and Export
Data Import with the LOAD TABLE Statement
Supported Character Sets
UNLOAD Statement [page 1453]
progress_messages Option
-sf Database Server Option
allow_read_client_file Option
-gl Database Server Option
on_charset_conversion_failure Option
Support for ESRI Shapefiles
-im Database Server Option
ClientFileValidator (CFV) Connection Parameter

1.4.4.201 LOCK FEATURE Statement

Prevents other concurrent connections from using a database server feature.

Syntax

```
LOCK FEATURE feature-name { ON | OFF }
```

```
feature-name :
'synchronization schema'
| 'all'
```

Parameters

feature-name

The name of the feature to be locked or unlocked. Specify all to unlock all the features locked by a connection.

ON | OFF

Specify ON to prevent other connections from using the feature. Specify OFF to allow connections to use the feature.

Remarks

You cannot lock a feature more than once for the same connection. If you attempt to unlock a feature that is not locked by the current connection and you do not specify all as the feature name, an error is returned. When a feature is locked by two or more connections, the feature must be unlocked by all connections before it can be used by other connections. Feature locks created by a connection are removed when the connection is dropped. Feature locks are removed when the database server is shut down.

When the synchronization schema feature is locked, the following statements cannot be executed by other connections:

- START SYNCHRONIZATION SCHEMA CHANGE
- CREATE SYNCHRONIZATION SUBSCRIPTION
- DROP SYNCHRONIZATION SUBSCRIPTION
- ALTER SYNCHRONIZATION SUBSCRIPTION
- ALTER PUBLICATION

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement prevents other connections from using the synchronization schema feature:

```
LOCK FEATURE 'synchronization schema' ON;
```

Related Information

LOCK TABLE Statement [page 1267]

1.4.4.202 LOCK MUTEX Statement

Locks a resource such as a file or system procedure using a predefined mutex.

Syntax

```
LOCK MUTEX [ owner.]mutex-name
[ IN { SHARE | EXCLUSIVE } MODE ]
[ TIMEOUT num-milliseconds ]
```

Parameters

owner

The owner of the mutex. `owner` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

mutex-name

The name of the mutex. `mutex-name` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

IN { SHARE | EXCLUSIVE } MODE clause

Use this clause to specify whether the lock provides exclusive access to the resource (EXCLUSIVE), or whether other connections can use the resource as well (SHARE). If the IN...MODE clause is not specified, then EXCLUSIVE is the default behavior.

TIMEOUT clause

The amount of time, in milliseconds (greater than 0), to wait to acquire the lock. If the TIMEOUT clause is not specified, then the connection waits indefinitely until the lock can be acquired.

`number-milliseconds` can be specified using a variable (for example, **`TIMEOUT @timeout-value`**). If `number-milliseconds` is set to a variable and the variable is NULL, the behavior is equivalent to not specifying the clause.

Remarks

Recursive LOCK MUTEX statements are allowed; however, an equal number of releases (RELEASE MUTEX) are required to release the mutex for connection-scope mutexes.

If a connection executes the LOCK MUTEX statement in SHARE MODE, and then again in EXCLUSIVE MODE, it may be blocked if other connections have the mutex locked in SHARE MODE. If not, then the lock mode changes to an exclusive lock and remains that way until the lock is completely released by the connection.

For transaction-scope mutexes (that is, the SCOPE TRANSACTION clause was specified at creation time), the mutex is held until the end of the transaction. For connection-scope mutexes (that is, the SCOPE CONNECTION clause was specified at creation time), the mutex is held until a RELEASE MUTEX statement is executed, or the connection is terminated.

LOCK MUTEX statements benefit from the same deadlock detection used for table and row locks.

Privileges

You must have the UPDATE ANY MUTEX SEMAPHORE system privilege or be the owner of the semaphore.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement locks the protect_my_cr_section mutex in exclusive mode:

```
LOCK MUTEX protect_my_cr_section IN EXCLUSIVE MODE;
```

Related Information

Mutexes and Semaphores
CREATE MUTEX Statement [page 906]
DROP MUTEX Statement [page 1107]
RELEASE MUTEX Statement [page 1329]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.203 LOCK TABLE Statement

Prevents other concurrent transactions from accessing or modifying a table.

Syntax

```
LOCK TABLE table-name
[ WITH HOLD ]
IN { SHARE | EXCLUSIVE } MODE
```

Parameters

table-name

The name of the table. The table must be a base table, not a view. As temporary table data is local to the current connection, locking global or local temporary tables has no effect.

WITH HOLD clause

Specify this clause to lock the table until the end of the connection. If the clause is not specified, the lock is released when the current transaction is committed or rolled back.

IN SHARE MODE clause

Specify this clause to obtain a shared table lock on the table, preventing other transactions from modifying the table but allowing them read access. If a transaction puts a shared lock on a table, it can change data in

the table provided no other transaction holds a lock of any kind on the row(s) being modified. Read locks on individual rows are not acquired when the IN SHARE MODE clause is selected.

IN EXCLUSIVE MODE clause

Specify this clause to obtain an exclusive table lock on the table, preventing other transactions from accessing the table. No other transaction can execute queries, updates, or any other action against the table. If a table is locked exclusively with a statement such as `LOCK TABLE...IN EXCLUSIVE MODE`, the default behavior is to not acquire row locks for the table. This behavior can be disabled by setting the subsume_row_locks option to Off.

Remarks

The LOCK TABLE statement allows direct control over concurrency at a table level, independent of the current isolation level.

While the isolation level of a transaction generally governs the kinds of locks that are set when the current transaction executes a request, the LOCK TABLE statement allows more explicit control locking of the rows in a table.

You cannot execute the LOCK TABLE statement on a view. However, if you execute the LOCK TABLE statement on a base table, a shared schema lock is created, which locks dependent views. A shared schema lock persists until the transaction is committed or rolled back.

Privileges

To lock a table in SHARE mode, you must be the owner of the table, or have SELECT privilege on the table, or have the SELECT ANY TABLE system privilege.

To lock a table in EXCLUSIVE mode, one of the following must be true:

- you are the table owner
- you have ALTER object-level privileges on the table
- you have the ALTER ANY TABLE system privilege
- you have the ALTER ANY OBJECT system privilege.

Side Effects

Other transactions that require access to the locked table may be delayed or blocked.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement prevents other transactions from modifying the Customers table for the duration of the current transaction:

```
LOCK TABLE GROUPO.Customers
IN SHARE MODE;
```

Related Information

Table Locks
How Locking Works
SELECT Statement [page 1362]
sa_locks System Procedure [page 1633]

1.4.4.204 LOOP Statement

Repeats the execution of a statement list.

Syntax

```
  [ statement-label : ]
[ WHILE search-condition ] LOOP
      statement-list
END LOOP [ statement-label ]
```

Remarks

The WHILE and LOOP statements are control statements that allow you to execute a list of SQL statements repeatedly while a `search-condition` evaluates to TRUE. The LEAVE statement can be used to resume execution at the first statement after the END LOOP.

If the ending `statement-label` is specified, it must match the beginning `statement-label`.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The LOOP/END LOOP statement is part of optional ANSI/ISO SQL Language Feature P002, "Computational completeness". In the ANSI/ISO SQL Standard, the WHILE DO/END WHILE statement is a separate statement that is also part of Language Feature P002. The syntax combination WHILE `search-condition` LOOP supported in the software is not in the standard.

Transact-SQL

LOOP is not supported in the Transact-SQL dialect. Looping within Transact-SQL stored procedures is done with the Transact-SQL WHILE statement.

Example

The following example fragment shows a WHILE loop in a procedure.

```
...
SET i = 1;
WHILE i <= 10 LOOP
   INSERT INTO Counters( number ) VALUES ( i );
   SET i = i + 1;
END LOOP;
...
```

The following example fragment shows a labeled LOOP in a procedure.

```
SET i = 1;
lbl:
LOOP
   INSERT
   INTO Counters( number )
   VALUES ( i );
   IF i >= 10 THEN
      LEAVE lbl;
   END IF;
   SET i = i + 1;
END LOOP lbl
```

Related Information

FOR Statement [page 1166]
CONTINUE Statement [page 818]

WHILE Statement [T-SQL] [page 1487]

1.4.4.205 MERGE Statement

Merges tables, views, and procedure results into a table or view.

Syntax

```
MERGE
INTO target-object [ into-column-list ]
USING [ WITH AUTO NAME ] source-object
   ON merge-search-condition
merge-operation [...]
 [ OPTION ( query-hint, ... ) ]
```

```
target-object :
[ userid.]target-table-name [ [ AS ] target-correlation-name ]
| [ userid.]target-view-name [ [ AS ] target-correlation-name ]
| ( select-statement ) [ AS ] target-correlation-name
```

```
source-object :
[ userid.]source-table-name [ [ AS ] source-correlation-name ] [ WITH ( table-
hints ) ]
| [ userid.]source-view-name [ [ AS ] source-correlation-name ]
| [ userid.]source-mat-view-name [ [ AS ] source-correlation-name ]
| ( select-statement ) [ AS ] source-correlation-name [  using-column-list ]
| procedure
```

```
procedure :
[ owner.]procedure-name ( procedure-syntax )
   [ WITH ( column-name data-type, ... ) ]
   [ [ AS ] source-correlation-name ]
```

```
merge-search-condition :
search-condition
| PRIMARY KEY
```

```
merge-operation :
WHEN MATCHED [ AND search-condition ] THEN match-action
| WHEN NOT MATCHED [ AND search-condition ] THEN not-match-action
```

```
match-action :
DELETE
| RAISERROR [ error-number ]
| SKIP
| UPDATE SET set-item, ...
| UPDATE [ DEFAULTS { ON | OFF } ]
```

```
not-match-action :
INSERT
| INSERT [ insert-column-list ] VALUES ( value, ... )
| RAISERROR [ error-number ]
| SKIP
```

```
set-item :
```

```
[target-correlation-name.]column-name = { expression | DEFAULT }
| [ owner-name.]target-table-name.column-name = { expression | DEFAULT }

insert-column-list :
( column-name, ... )

query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| option-name = option-value

into-column-list :
( column-name, ... )

using-column-list :
( column-name, ... )

error-number : positive integer or variable greater than 17000

option-name : identifier

option-value :
hostvar (indicator allowed)
| string
| identifier
| number

table-hints : see the documentation for the FROM clause

search-condition : see the documentation for search conditions

set-clause-list : see the documentation for the SET statement
```

Parameters

INTO clause

Use this clause to define the target object for the MERGE statement. `target-object` can be the name of a base table, regular view, or derived table; it cannot be a materialized view. The derived table or view must represent an updatable query block. For example, if the view or derived table definition contains UNION, INTERSECT, EXCEPT, or GROUP BY, then it cannot be used as a target object for the MERGE statement.

When `target-object` is a derived table, the optional `into-column-list` can be used to provide alternate names for the columns of the derived table. When used in this manner, the size of the `into-column-list` must match the column list for the derived table, and the ordering of the two lists must be the same.

When `target-object` is a base table or view, `into-column-list` can be used to specify a subset of the table or view columns as relevant for the rest of the MERGE statement.

The database server uses into-column-list to resolve:

- UPDATE without a SET clause in WHEN MATCHED clause
- INSERT without a VALUES clause in a WHEN NOT MATCHED clause
- PRIMARY KEY search condition in the ON clause
- WITH AUTO NAME clause in the USING clause

If you do not specify into-column-list, then into-column-list is assumed to contain all the columns of the target-object.

USING clause

Use this clause to define the source of the data you are merging from. source-object can be a base table (including table hints), a view, a materialized view, a derived table, or a procedure. If source-object is a derived table, you can specify using-column-list. All columns of source-object are used if you do not specify using-column-list.

WITH AUTO NAME clause

Use this clause to get the server to automatically use column names to match columns in the into-column-list columns in target-object for the merge operation. The following examples are equivalent and demonstrate that the order of the columns in into-column-list changes to match the names of the columns in the source-object when WITH AUTO Name is specified:

```
... INTO T ( Name, ID, Description )
   USING WITH AUTO NAME ( SELECT Description, Name, ID FROM Products WHERE
Description LIKE '%cap%')
... INTO T ( Description, Name, ID )
   USING ( SELECT Description, Name, ID FROM Products WHERE Description LIKE
'%cap%' )
```

ON clause

Use this clause to specify the condition to match a row in source-object with rows in target-object.

You can specify ON PRIMARY KEY to match source-object rows based on the target-object primary key definition. source-object does not need a primary key. However, target-object must have a primary key. When specifying ON PRIMARY KEY:

- An error is returned if target-object is not a base table, or if it does not have a primary key.
- An error is returned if one or more primary key columns are not included in into-column-list.
- The number of columns in into-column-list and using-column-list can be different as long as every primary key column in into-column-list has a corresponding matching column in using-column-list. For example, if into-column-list is (I1, I2, I3), using-column-list is (U1, U2), and the primary key columns are (I2, I3), an error is returned because column (I3) of the target-object primary key does not have a match in the using-column-list.
- Regardless of the definition of the primary key, matching of primary key columns in into-column-list to expressions in using-column-list is based on the position of the primary key columns in into-column-list. For example, suppose the primary key on target-object is defined as (B, C), and the into-column-list is (E, C, F, A, D, B). When ON PRIMARY KEY is specified, target-object column B is compared to the sixth element of using-column-list because column B is in the sixth position in the into-column-list. Likewise, target-object column C is compared to the second element of using-column-list.
  ON PRIMARY KEY is syntactic shorthand for a corresponding ON condition. For example, assume that into-column-list is (I1, I2, .. I_n), and that the corresponding matched using-column-list is (U1,

U2, .. Um). Also assume that the primary key columns of target-object are I1, I2, I3 and all the primary key columns are contained in into-column-list. In this case, merge-search-condition is defined as the conjunct "I1=U1 AND I2=U2 AND I3=U3".

WHEN MATCHED and WHEN NOT MATCHED clauses

Use the WHEN MATCHED and WHEN NOT MATCHED clauses to define an action to take when a row from source-object matches or does not match a row in target-object. You specify the action after the THEN keyword. You can control the actions to take for subsets of matching or non-matching rows by specifying an additional AND clause.

The ON clause determines how rows from source-object are separated into matching and non-matching rows. A row in source-object is considered a matching row when the ON clause is TRUE for at least one row in target-object. A row from source-object is considered a non-matching row when the ON clause is not TRUE for any rows in target-object. Use multiple WHEN MATCHED and WHEN NOT MATCHED clauses to partition sets of matching and non-matching rows into disjoint subsets. Each subset is processed by a WHEN clause. WHEN MATCHED and WHEN NOT MATCHED clauses are processed in the order they appear in the MERGE statement.

The search condition specified in the AND clause of a WHEN MATCHED or WHEN NOT MATCHED clause determines if a candidate row is processed by the specific clause. When you specify a WHEN MATCHED or WHEN NOT MATCHED clause without the AND clause the search condition in the AND clause is assumed to be TRUE. If a row satisfies the AND condition for more than one clause, the row is processed by the clause that appears first in the MERGE statement.

An error is returned when any of the WHEN MATCHED clauses process the same target-object row more than once. A target-object row can belong to the same subset of the same WHEN MATCHED clause more than once if it matches two different input rows from the source-object.

In the following example an error is returned because the row with ID 300 from the target-object Products matches 111 rows from the source-object SalesOrderItems. All the matches belong to the same subset corresponding to the WHEN MATCHED THEN UPDATE clause.

```
MERGE INTO GROUPO.Products
   USING GROUPO.SalesOrderItems S
   ON S.ProductID = Products.ID
   WHEN MATCHED THEN UPDATE SET Products.Quantity = 20;
```

WHEN MATCHED: For a matching row, you can specify one of the following actions for match-action:

DELETE

Specify DELETE to delete the row from target-object.

RAISERROR

Specify RAISERROR to terminate the merge operation, roll back any changes, and return an error. By default, when you specify RAISERROR, the database server returns SQLSTATE 23510 and SQLCODE -1254.

Optionally, you can customize the SQLCODE that is returned by specifying the error-number parameter after the RAISERROR keyword. The custom SQLCODE must be a positive integer greater than 17000, and can be specified either as a number or a variable. When you specify a custom SQLCODE, the number returned is a negative number.

For example, if you specify `WHEN MATCHED AND search-condition THEN RAISERROR 17001`, then, when a row is found that satisfies the conditions of the WHEN clause, the merge operation fails, changes are rolled back, and the error returned has SQLSTATE 23510 and SQLCODE -17001.

SKIP

Specify SKIP to skip the row; no action is taken.

UPDATE

Specify UPDATE SET to update the row using the `set-item` values. `set-item` is a simple assignment expression where a column is set to the value of `expression`. There are no restrictions on the `expression`. You can also specify DEFAULT to set the column to the default defined for the column.

For example, `UPDATE SET target-column1=DEFAULT, target-column2=source-column2` sets target-column1 to its default value and sets target-column2 to be the same as the modify row from source-column2 in `source-object`.

If you do not specify the SET clause, the SET clause is defined by `into-column-list` and `using-column-list`. For example, if `into-column-list` is (I1, I2, .. In), and `using-column-list` is (U1, U2, .. Un) the SET clause is assumed to be "`SET I1=U1 , I2=U2 , .. In=Un`".

WHEN NOT MATCHED: For a non-matching row, you can specify one of the following actions for `non-match-action`:

INSERT

Specify INSERT...VALUES to insert the row using the specified values. When you specify the INSERT clause without a VALUES clause, the VALUES clause is defined by `into-column-list` and `using-column-list`. For example, if `into-column-list` is (I1, I2, .. In), and `using-column-list` is (U1, U2, .. Un), the INSERT without a VALUES clause is equivalent to `INSERT (I1, I2, .. In) VALUES (U1, U2, .. Un)`.

RAISERROR

Specify RAISERROR to terminate the merge operation, roll back any changes, and return an error. When you specify RAISERROR, the database server returns SQLSTATE 23510 and SQLCODE -1254 by default. Optionally, you can customize the SQLCODE that is returned by specifying the `error-number` parameter after the RAISERROR keyword. The custom SQLCODE must be a positive integer greater than 17000, and can be specified either as a number or a variable. When you specify a custom SQLCODE, the number returned is a negative number.

For example, if you specify `WHEN NOT MATCHED AND search-condition THEN RAISERROR 17001`, then, when a row is found that satisfies the conditions of the WHEN clause, the merge operation fails, changes are rolled back, and the error returned has SQLSTATE 23510 and SQLCODE -17001.

SKIP

Specify SKIP to skip the row; no action is taken.

OPTION clause

Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- `option-name` = `option-value`. A `OPTION( isolation_level = ... )` specification in the query text overrides all other means of specifying isolation level for a query.

Remarks

Rows in `source-object` are compared to rows in `target-object` and found to be matching or non-matching depending on whether they satisfy the conditions of the ON clause. Rows in `source-object` are considered a match if there exists at least one row in `target-table` such that `merge-search-condition` evaluates to true. Matching rows and non-matching rows are then grouped by the actions defined for them in the WHEN MATCHED and WHEN NOT MATCHED clauses according to the search conditions specified by the AND clauses. The process of grouping rows by matched and non-matched actions is referred to as **branching**, and each group is referred to as a **branch**.

Once branching is complete, the database begins executing the action defined for the rows of the branch. Branches are processed in the order in which they occur, which matches the order in which the WHEN clauses occur in the statement. If, during branching, more than one row in `source-object` has an action defined for the same row in `target-object`, the merge operation fails and an error is returned. This prevents the merge operation from performing more than one action on any given row in `target-object`.

As branches are processed, the insert, update, and delete actions are recorded in the transaction log as their respective INSERT, UPDATE, and DELETE statements.

Privileges

Required privileges are determined at execution time and depend on the objects specified, and the operations the merge would result in:

Selecting source-object or target-object

You must be the owner of the objects, or have the SELECT ANY TABLE system privilege, or have SELECT privilege on the target object.

Inserting rows into target-object

You must be the owner of `target-object`, or have the INSERT ANY TABLE system privilege, or have INSERT privilege on the target object.

Updating rows in target-object

You must be the owner of `target-object`, or have the UPDATE ANY TABLE system privilege, or have UPDATE privilege on the target object.

Deleting rows from target-object

You must be the owner of `target-object`, or have the DELETE ANY TABLE system privilege, or have DELETE privilege on the target object.

EXECUTE privilege is required on any procedure referenced in the MERGE statement.

Side Effects

Any triggers defined for `target-object` are fired.

Standards

ANSI/ISO SQL Standard

The MERGE statement comprises Features F312 and F313 of the ANSI/ISO SQL Standard. The MERGE statement in the software is compliant with the MERGE statement specification in the ANSI/ISO SQL Standard, with additional extensions. The software-specific extensions to the MERGE statement include:

- DELETE in a WHEN MATCHED clause
- RAISERROR in a WHEN [NOT] MATCHED clause
- SKIP in a WHEN [NOT] MATCHED clause
- OPTION clause
- PRIMARY KEY clause
- DEFAULTS clause
- INSERT clause without a VALUES clause
- WITH AUTO NAME clause
- UPDATE clause without the SET clause

Example

The following example merges a row from a derived table into the Products table, effectively adding a new tee shirt with the same attributes as an existing tee shirt, but with a new color, quantity, and product identifier. In this example if the product with identification number 304 already exists in the Products table then the row is not inserted:

```
MERGE INTO Products ( ID, Name, Description, Size, Color, Quantity, UnitPrice,
Photo )
   USING WITH AUTO NAME (
      SELECT 304 AS ID,
             'Purple' AS Color,
             100 AS Quantity,
             Name,
             Description,
             Size,
             UnitPrice,
             Photo
          FROM Products WHERE Products.ID = 300 ) AS DT
   ON PRIMARY KEY
   WHEN NOT MATCHED THEN INSERT;
```

Related Information

Data Import with the MERGE Statement
FROM Clause [page 1173]
Search Conditions [page 58]
SET Statement [page 1373]
UPDATE Statement [page 1463]

INSERT Statement [page 1232]
DELETE Statement [page 1070]
SELECT Statement [page 1362]
Search Conditions [page 58]
MERGE statement for table '%1' failed because of a RAISERROR specification in the statement

1.4.4.206 MESSAGE Statement

Displays a message.

Syntax

```
MESSAGE expression
[ TYPE { INFO | ACTION | WARNING | STATUS } ]
[ TO { CONSOLE
    | CLIENT [ FOR { CONNECTION conn-id-number [ IMMEDIATE ] | ALL } ]
    | [ EVENT | SYSTEM ] LOG }
    [ DEBUG ONLY ] ]
```

```
conn-id : integer
```

Parameters

TYPE clause

This clause specifies the message type. The client application must decide how to handle the message. For example, if you specify TO CLIENT, Interactive SQL displays messages in the following locations:

INFO

The *History* tab. INFO is the default type.

ACTION

A window with an *OK* button.

WARNING

A window with an *OK* button.

STATUS

The *History* tab.

TO clause

This clause specifies the destination of a message:

CONSOLE

Send messages to the database server messages window and the database server message log file if one has been specified. CONSOLE is the default.

CLIENT

Send messages to the client application. Your application must decide how to handle the message, and you can use the TYPE as information on which to base that decision.

LOG

Send messages to the database server message log file specified by the -o option. If EVENT or SYSTEM is specified, the message is also written to the database server messages window and to the event log. Messages in the database server message log are identified as follows:

i

Messages of type INFO or STATUS.

w

Messages of type WARNING.

e

Messages of type ACTION.

FOR clause

For messages TO CLIENT, this clause specifies which connections receive notification about the message. By default, the connection receives the message the next time a SQL statement or a WAITFOR DELAY statement is executed.

CONNECTION conn-id-number

Specify the recipient's connection ID number. If IMMEDIATE is specified, the connection receives the message within a few seconds regardless of when the SQL statement is executed.

ALL

Specify that all open connections receive the message.

DEBUG ONLY

This clause allows you to control whether debugging messages added to stored procedures and triggers are enabled or disabled by changing the setting of the debug_messages option. When DEBUG ONLY is specified, the MESSAGE statement is executed only when the debug_messages option is set to On.

> **i** Note
>
> DEBUG ONLY messages are inexpensive when the debug_messages option is set to Off, so these statements can usually be left in stored procedures on a production system. However, they should be used sparingly in locations where they would be executed frequently; otherwise, they may result in a small performance penalty.

Remarks

The MESSAGE statement displays a message, which can be any expression. Clauses can specify the message type and where the message appears.

If the size of `expression` exceeds the database page size, `expression` is truncated to fit within the database page size. To check the page size in effect for the database, you can query the PageSize database property (`SELECT DB_PROPERTY( 'PageSize' );`).

The procedure executing a MESSAGE...TO CLIENT statement must be associated with a connection.

For example, the window is not displayed in the following example because the event occurs outside a connection.

```
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
   MESSAGE 'Idle server' TYPE WARNING TO CLIENT;
END;
```

However, in the following example, the message is written to the database server messages window.

```
CREATE EVENT CheckIdleTime2
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
   MESSAGE 'Idle server' TYPE WARNING TO CONSOLE;
END;
```

Valid expressions can include a quoted string or other constant, variable, or function.

The FOR clause can be used to notify another application of an event detected on the database server without the need for the application to explicitly check for the event. When the FOR clause is used, recipients receive the message the next time that they execute a SQL statement. If the recipient is currently executing a SQL statement, the message is received when the statement completes. If the statement being executed is a stored procedure call, the message is received before the call is completed.

If an application requires notification within a short time after the message is sent and when the connection is not executing SQL statements, use the IMMEDIATE clause to implement client notification and not multiple concurrent WAITFOR DELAY statements.

Typically, messages sent using the IMMEDIATE clause are delivered in less than five seconds, even if the destination connection is not making database server requests. Message delivery could be delayed if the client connection makes several requests per second, receives very large BLOB data, or if the client's message callback executes for more than a second. In addition, sending more than one IMMEDIATE message to a single connection every two seconds could delay message delivery or generate an error message. If the client connection is disconnected, a successful MESSAGE...IMMEDIATE statement may not be delivered.

Messages sent without the IMMEDIATE clause are only delivered when the client executes a specific request, or a WAITFOR DELAY statement. As a result, the delivery time of messages is unlimited.

The IMMEDIATE clause requires a SQL Anywhere 11 or later DBLib, ODBC, or SQL Anywhere JDBC driver. The IMMEDIATE clause is not supported by non-threaded UNIX and Linux client libraries. An error message is generated when a message is sent to a destination connection that does not support the IMMEDIATE clause. An error message is generated when an IMMEDIATE message is sent to the same connection executing the MESSAGE statement.

```
MESSAGE 'Please disconnect' TYPE WARNING TO CLIENT
    FOR CONNECTION 16 IMMEDIATE;
```

A MESSAGE...TO CLIENT expression can be truncated to 2048 bytes. For messages sent with the IMMEDIATE clause, the message expression can be truncated to the smaller of the packet size of the connection or 2048 bytes.

Embedded SQL and ODBC clients receive messages via message callback functions. In each case, these functions must be registered. In Embedded SQL, the message callback is registered with db_register_a_callback using the DB_CALLBACK_MESSAGE parameter. In ODBC, the message callback is registered with SQLSetConnectAttr using the SA_REGISTER_MESSAGE_CALLBACK parameter.

Privileges

To execute a MESSAGE statement containing a FOR clause, a TO EVENT LOG clause, or a TO SYSTEM LOG clause, you must have the SERVER OPERATOR system privilege. Otherwise, no privileges are required for this statement.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following procedure displays a message in the database server messages window:

```
CREATE PROCEDURE message_text( )
BEGIN
MESSAGE 'The current date and time: ', Now( );
END;
```

The following statement displays the string `The current date and time`, followed by the current date and time, in the database server messages window.

```
CALL message_text( );
```

Related Information

Formatting Event Log Messages
sa_conn_info System Procedure [page 1547]

CREATE PROCEDURE Statement [page 913]
debug_messages Option
db_register_a_callback Function
WAITFOR Statement [page 1481]

1.4.4.207 NOTIFY SEMAPHORE Statement

Increments the counter associated with a semaphore.

Syntax

```
NOTIFY SEMAPHORE [owner.]semaphore-name
[ INCREMENT BY number ]
```

Parameters

owner

The owner of the semaphore. `owner` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

semaphore-name

The name of the semaphore. `semaphore-name` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

INCREMENT BY clause

Specify a positive integer to indicate how much to increment the counter associated with the semaphore. If this clause is not specified, then the counter is incremented by 1.

`number` can be specified using a variable (for example, **`INCREMENT BY @inc-number`**).

If you set `number` to NULL, or if it is set to a variable and the variable value is NULL, the behavior is equivalent to not specifying the clause.

Remarks

If the counter is currently 0 at the time it is incremented, and one or more connections are blocked on a WAITFOR SEMAPHORE statement that uses this semaphore, the NOTIFY SEMAPHORE statement notifies each of the connections. Each blocked connection, in turn, unblocks if the counter value is not zero and decrements the counter value. If the counter value is decremented to zero, then the next blocked connection continues to block. Blocked connections are unblocked in first-in, first-out (FIFO) order.

If a connection that notified a semaphore is dropped or canceled, the counter increment persists.

Privileges

You must have the UPDATE ANY MUTEX SEMAPHORE system privilege or be the owner of the semaphore.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a semaphore and sets its initial value to zero:

```
CREATE OR REPLACE SEMAPHORE DBA.gate START WITH 0;
```

The following statements define a stored procedure that waits on this semaphore. If the semaphore counter is not zero, it will decrement the counter and proceed. If the semaphore counter is zero, it wil block:

```
CREATE OR REPLACE PROCEDURE SemTest()
BEGIN
    WAITFOR SEMAPHORE DBA.gate;
    SELECT 'Waitfor done';
END;
```

The following statement calls the stored procedure. Execution will be delayed since the semaphore is initially zero and the stored procedure blocks. It will block until a NOTIFY SEMAPHORE statement is executed on another connection.

```
CALL SemTest;
```

Execute the following statement on a separate connection. It increments the semaphore by 1 and allows a blocked connection to proceed. If no connection is blocked, the next WAITFOR statement that references the semaphore will proceed without blocking.

```
NOTIFY SEMAPHORE DBA.gate INCREMENT BY 1;
```

Execute the following statement on a separate connection. It increments the semaphore by 2 and allows two blocked connections to proceed. If only one connection is blocked, the next WAITFOR statement that

references the semaphore will proceed without blocking. If no connections are blocked, the next 2 WAITFOR statements that reference the semaphore will proceed without blocking.

```
NOTIFY SEMAPHORE DBA.gate INCREMENT BY 2;
```

Related Information

Mutexes and Semaphores
CREATE SEMAPHORE Statement [page 957]
DROP SEMAPHORE Statement [page 1118]
WAITFOR SEMAPHORE Statement [page 1483]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.208 NOTIFY TRACE EVENT Statement

Logs a user-defined trace event to a trace session.

Syntax

```
NOTIFY TRACE EVENT trace-event-name ( [ param1 [ ,... ] ] )
```

Parameters

trace-event-name

The trace event name must be the name of a user-defined trace event. It cannot be a system-defined trace event.

param1

The values of the trace event fields.

Remarks

This statement is used to notify any sessions that include the specified trace event. If a trace event is not being traced by any session, then this statement has no effect and the parameters are not evaluated (for example, by a call to a user-defined function).

System privileges

You must have the NOTIFY TRACE EVENT system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statements log events to the current (fictitious) event trace session, my_event.

```
NOTIFY TRACE EVENT my_event( 1, 'Hello world' ); -- trigger user-defined trace
events
NOTIFY TRACE EVENT my_event( 2, 'Hello world 2' );
NOTIFY TRACE EVENT my_event( 3, 'Hello world 3' );
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.4.4.209 OPEN Statement [ESQL] [SP]

Opens a previously declared cursor to access information from the database.

Syntax

Embedded SQL

```
OPEN cursor-name
[ USING { DESCRIPTOR sqlda-name | hostvar, ... } ]
[ WITH HOLD ]
[ ISOLATION LEVEL isolation-level ]
[ BLOCK n ]
```

Stored procedures

```
OPEN cursor-name
[ WITH HOLD ]
[ ISOLATION LEVEL isolation-level ]
```

```
cursor-name : identifier | hostvar
```

```
sqlda-name : identifier
```

```
isolation-level : 0 | 1 | 2 | 3 | SNAPSHOT | STATEMENT SNAPSHOT |
READONLY STATEMENT SNAPSHOT
```

Parameters

USING DESCRIPTOR clause

The USING DESCRIPTOR clause is for Embedded SQL only. It specifies the host variables to be bound to the placeholder bind variables in the SELECT statement for which the cursor has been declared.

OPEN...USING cannot be used in a stored procedure.

WITH HOLD clause

By default, all cursors are automatically closed at the end of the current transaction (COMMIT or ROLLBACK). The optional WITH HOLD clause keeps the cursor open for subsequent transactions. It remains open until the end of the current connection or until an explicit CLOSE statement is executed. Cursors are automatically closed when a connection is terminated.

Upon COMMIT or ROLLBACK, all long-term row locks held by the connection are released, including those rows that constitute the result set of a WITH HOLD cursor. However, cursor stability locks, which are acquired at isolation levels 1, 2, and 3, are retained for the life of the cursor and are only released when the cursor is closed or when the connection terminates.

Upon completion of a ROLLBACK statement, the contents of, and positioning within, a WITH HOLD cursor are unpredictable and are not guaranteed. You can use the ansi_close_cursors_on_rollback option to control whether or not a ROLLBACK statement will close WITH HOLD cursors automatically.

ISOLATION LEVEL clause

The ISOLATION LEVEL clause allows this cursor to be opened at an isolation level different from the current setting of the isolation_level option. All operations on this cursor are performed at the specified isolation level regardless of the option setting. If this clause is not specified, then the cursor's isolation level for the entire time the cursor is open is the value of the isolation_level option when the cursor is opened.

The following values are supported:

- 0
- 1
- 2
- 3
- SNAPSHOT
- STATEMENT SNAPSHOT
- READONLY STATEMENT SNAPSHOT

The cursor is positioned before the first row.

BLOCK clause

This clause is for Embedded SQL use only. Rows may be fetched by the client application more than one at a time. This is referred to as block fetching, prefetching, or multi-row fetching. The BLOCK clause can reduce the number of rows prefetched. Specifying the BLOCK clause on OPEN is the same as specifying the BLOCK clause on each FETCH.

Remarks

The OPEN statement opens the named cursor. The cursor must be previously declared.

The OPEN statement may return a SQL warning if the cursor type does not match the characteristics of the cursor's underlying statement.

When the cursor is on a CALL statement, OPEN causes the procedure to execute until the first result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set.

Embedded SQL usage

After successful execution of the OPEN statement, the `sqlerrd[3]` field of the SQLCA (SQLIOESTIMATE) is filled in with an estimate of the number of input/output operations required to fetch all rows of the query. Also, the `sqlerrd[2]` field of the SQLCA (SQLCOUNT) is filled with either the actual number of rows in the cursor (a value greater than or equal to 0), or an estimate thereof (a negative number whose absolute value is the estimate). It is the actual number of rows if the database server can compute it without counting the rows. The database can also be configured to always return the actual number of rows, but this can be expensive.

If `cursor-name` is specified by an identifier or string, the corresponding DECLARE CURSOR statement must appear before the OPEN in the C program; if the `cursor-name` is specified by a host variable, the DECLARE CURSOR statement must execute before the OPEN statement.

Privileges

When the cursor is on a SELECT statement, you must be the owner of the object referenced in the cursor, or have SELECT privilege on the object, or have the appropriate SELECT system privilege (for example, SELECT ANY TABLE).

When the cursor is on a CALL statement, you must be the owner of the procedure or have EXECUTE privilege on the procedure, or have the EXECUTE ANY PROCEDURE system privilege.

Side Effects

OPEN causes the complete materialization of an INSENSITIVE cursor's result set.

If access plan caching is enabled, some SQL warnings that would be returned to the application at OPEN time may be suppressed. The suppressed warnings include warnings to indicate that the cursor type has changed, that the underlying query is not deterministic, or that string truncation has occurred with one or more literal constants embedded in the statement.

Standards

ANSI/ISO SQL Standard

Use of the OPEN statement within Embedded SQL is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The use of the OPEN statement within a stored procedure is a Core Feature. The ISOLATION LEVEL and BLOCK clauses are not in the standard, as is the ability to OPEN a cursor over a CALL statement. In the ANSI/ISO SQL Standard, WITH HOLD is specified as part of the DECLARE CURSOR statement, and not on OPEN.

The setting of specific values in the SQLCA is not in the standard.

Transact-SQL

The OPEN statement is supported by Adaptive Server Enterprise. Adaptive Server Enterprise does not support the ISOLATION LEVEL, BLOCK, and WITH HOLD clauses.

Example

The following examples show the use of OPEN in Embedded SQL.

```
EXEC SQL OPEN employee_cursor;
```

```
EXEC SQL PREPARE emp_stat FROM
'SELECT empnum, empname FROM GROUPO.Employees WHERE name like ?';
EXEC SQL DECLARE employee_cursor CURSOR FOR emp_stat;
EXEC SQL OPEN employee_cursor USING :pattern;
```

This example fragment shows an OPEN statement in a procedure or trigger.

```
BEGIN
   DECLARE cur_employee CURSOR FOR
   SELECT Surname
   FROM GROUPO.Employees;
   DECLARE name CHAR(40);
   OPEN cur_employee;
   LP: LOOP
         FETCH NEXT cur_employee INTO name;
         IF SQLCODE <> 0 THEN LEAVE LP END IF;
    ...
   END LOOP
   CLOSE cur_employee;
END
```

Related Information

Plan Caching
Insensitive Cursors
Cursors in Embedded SQL
Cursors in Procedures, Triggers, User-defined Functions, and Batches
How Locking Works
Lock Duration
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
RESUME Statement [page 1341]
PREPARE Statement [ESQL] [page 1308]
FETCH Statement [ESQL] [SP] [page 1162]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
RESUME Statement [page 1341]
CLOSE statement [ESQL] [SP] [page 807]
FOR Statement [page 1166]
ansi_close_cursors_on_rollback Option
row_counts Option
close_on_endtrans Option

1.4.4.210 OUTPUT Statement [Interactive SQL]

Outputs the current query results to a file.

Syntax

Output to a file

```
OUTPUT TO filename
[ APPEND ]
[ BYTE ORDER MARK { ON | OFF }
[ COLUMN WIDTHS ( integer, ...) ]
[ DELIMITED BY string ]
```

```
[ ENCODING encoding ]
[ ESCAPE CHARACTER character ]
[ ESCAPES { ON | OFF } ]
[ FORMAT output-format ]
[ HEXADECIMAL { ON | OFF | ASIS } ]
[ QUOTE string [ ALL ] ]
[ VERBOSE ]
[ WITH COLUMN NAMES ]
```

```
output-format :
TEXT
| EXCEL
| FIXED
| HTML
| SQL
| XML
```

```
encoding : string | identifier
```

Output to an ODBC data source

```
OUTPUT
USING connection-string
INTO destination-table-name
[ CREATE TABLE { ON | OFF } ]
```

```
connection-string :
{ DSN = odbc-data-source
| DRIVER = odbc-driver-name [; connection-parameter = value [; ... ] ] }
```

Parameters

APPEND clause

This optional keyword is used to append the results of the query to the end of an existing output file without overwriting the previous contents of the file. If the APPEND clause is not used, the OUTPUT statement overwrites the contents of the output file by default. The APPEND keyword is valid if the output format is TEXT, FIXED, or SQL.

BCP FORMAT clause

The BCP format clause is used to import and export files between SQL Anywhere and Adaptive Server Enterprise.

BYTE ORDER MARK clause

Use this clause to specify whether to include a byte order mark (BOM) at the start of a Unicode file. By default, this option is ON, which directs Interactive SQL to write a byte order mark (BOM) at the beginning of the file. If BYTE ORDER MARK is OFF, DBISQL does not write a BOM.

The BYTE ORDER MARK clause is relevant only when writing TEXT formatted files. Attempts to use the BYTE ORDER MARK clause with FORMAT clauses other than TEXT returns an error.

The BYTE ORDER MARK clause is used only when reading or writing files encoded with UTF-8 or UTF-16 (and their variants). Attempts to use the BYTE ORDER MARK clause with any other encoding returns an error.

COLUMN WIDTHS clause

The COLUMN WIDTHS clause is used to specify the column widths for the FIXED format output.

CREATE TABLE clause

Use the CREATE TABLE clause to specify whether to create the destination table if it does not exist. The default is ON.

DELIMITED BY clause

The DELIMITED BY clause is for the TEXT output format only. The delimiter string is placed between columns. The delimited string is controlled by the isql_field_separator_option.

By default, the delimiter is a comma for locales that use a period as the decimal separator, and a semicolon for locales that use a comma as the decimal separator.

ENCODING clause

The ENCODING clause allows you to specify the encoding that is used to write the file. The ENCODING clause can only be used with the TEXT format.

The ENCODING clause is useful when you have data that cannot be represented in the operating system character set. In this case, if you do not use the ENCODING clause, characters that cannot be represented in the default encoding are lost in the output (that is, a lossy conversion occurs).

If the input file was created using the OUTPUT statement and an encoding was specified, then the same ENCODING clause should be specified on the INPUT statement.

When running Interactive SQL, the encoding that is used to export the data is determined in the following order:

- The encoding specified by the ENCODING clause (if this clause is specified)
- The encoding specified with the default_isql_encoding option (if this option is set).
- The default encoding for the platform you are running on. On English Windows computers, the default encoding is 1252.

ESCAPE CHARACTER clause

The default escape character for characters stored as hexadecimal codes and symbols is a backslash (\). For example, \x0A is the line feed character.

This setting can be changed using the ESCAPE CHARACTER clause. For example, to use the exclamation mark as the escape character, specify:

```
... ESCAPE CHARACTER '!'
```

The new line character can be specified as '\n'. Other characters can be specified using hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\ ) is interpreted as a single backslash. A backslash followed by any character other than n, x, X, or \ is interpreted as two separate characters. For example, \q is interpreted as a backslash and the letter q.

ESCAPES clause

With ESCAPES turned on (the default), characters following the backslash character are recognized and interpreted as special characters by the database server. With ESCAPES turned off, the characters are written exactly as they appear in the source data.

FORMAT clause

The FORMAT clause allows you to specify the file format for the output. If you do not specify the FORMAT clause, the format specified by the output_format option is used. If you specify the FORMAT clause, the

setting of the output_format option is ignored. The default output format is TEXT. Allowable output formats are:

TEXT

The output is a TEXT format file with one row per line in the file. All values are separated by a field delimiter, and strings are enclosed in apostrophes (single quotes). Field delimiters are typically commas (,) or semicolons (;). The delimiter and quote strings can be changed using the DELIMITED BY and QUOTE clauses.

Three other special sequences are also used. The two characters \n represent a newline character, \\ represents a single \, and the sequence \xDD represents the character with hexadecimal code DD.

To TEXT without including quotes or newlines in your output, turn off quotes and escapes as follows: `QUOTE '' ESCAPES OFF`.

EXCEL

When files with a `.csv` or `.txt` extension are exported with the FORMAT EXCEL clause, they follow the default formatting for Microsoft Excel files. If the file name does not have a `.txt` or `.csv` file extension, then the output is a Microsoft Excel worksheet with a `.xlsb` extension. Column headings are written to the first row on the sheet.

When exporting to a Microsoft Excel file, the Microsoft Excel ODBC driver must be installed, and the bitness of Interactive SQL and the Microsoft Excel ODBC driver must match.

Exporting to a Microsoft Excel workbook (`.xl*`) file is only supported on Windows.

FIXED

The output is fixed format with each column having a fixed width. The width for each column can be specified using the COLUMN WIDTHS clause. No column headings are output in this format.

If the COLUMN WIDTHS clause is omitted, the width for each column is computed from the data type for the column, and is large enough to hold any value of that data type. The exception is that LONG VARCHAR and LONG BINARY data default to 32 KB.

HTML

The output is in the Hyper Text Markup Language format.

SQL

The output is an Interactive SQL INPUT statement (required to recreate the information in the table) in a `.sql` file.

XML

The output is an XML file encoded in UTF-8 and containing an embedded DTD. Binary values are encoded in CDATA blocks with the binary data rendered as 2-hex-digit strings.

HEXADECIMAL clause

The HEXADECIMAL clause specifies how binary values are output for the TEXT format. Allowable values are:

ON

When set to ON, binary values are written with an 0x prefix followed by a series of hexadecimal pairs; for example, 0xabcd.

OFF

When set to OFF, unprintable character values are prefixed with the escape character, such as a backslash, followed by an x, and then followed by the hexadecimal pair for the byte. Printable characters are output as-is.

ASIS

When set to ASIS, values are written as is, without any escaping, even if the values contain control characters. ASIS is useful for text that contains formatting characters such as tabs or carriage returns.

QUOTE clause

The QUOTE clause is for the TEXT output format only. The quote string is placed around string values. The default is a single quote ('). If ALL is specified in the QUOTE clause, the quote string is placed around all values, not just around strings. To suppress quoting, specify empty single quotes. For example, `QUOTE ''`.

USING clause

The USING clause exports data to an ODBC data source. You can either specify the ODBC data source name with the DSN option, or the ODBC driver name and connection parameters with the DRIVER option. `Connection-parameter` is an optional list of database-specific connection parameters.

`odbc-data-source` is the name of a user or ODBC data source name. For example, `odbc-data-source` for the SQL Anywhere sample database is SQL Anywhere 17 Demo.

`odbc-driver-name` is the ODBC driver name. For a SQL Anywhere database, the `odbc-driver-name` is SQL Anywhere; for an UltraLite database, `odbc-driver-name` is UltraLite 17.

VERBOSE clause

When the optional VERBOSE keyword is included, error messages about the query, the SQL statement used to select the data, and the data itself are written to the output file. Lines that do not contain data are prefixed by two hyphens. If VERBOSE is omitted (the default) only the data is written to the file. The VERBOSE keyword is valid if the output format is TEXT, FIXED, or SQL.

WITH COLUMN NAMES clause

The WITH COLUMN NAMES clause inserts the column names in the first line of the text file. The WITH COLUMN NAMES clause is for TEXT format only. For Microsoft Excel files, the column headings are written to the first row in the file.

WORKSHEET

When the FORMAT clause is set to EXCEL, the WORKSHEET clause specifies the worksheet within the Microsoft Excel file that data is exported to. If the clause is omitted, then data is exported to a default worksheet named *Results*.

Remarks

The OUTPUT statement is used directly after a statement that retrieves the data to be output.

The OUTPUT statement with its clauses APPEND and VERBOSE is equivalent to the >#, >>#, >&, and >>& operators of earlier versions of Interactive SQL. These operators redirect data, but the Interactive SQL statements allow for more precise output and easier-to-read code.

If the entire result set has not been cached, then the OUTPUT statement re-executes the query. The entire result set is not cached if the number of rows in the result set exceeds the maximum value of the isql_maximum_displayed_rows option (500 by default).

If the executed statement returns multiple result sets, then Interactive SQL creates a file for each result set. The files are named `filename_x`, where `x` is a counter starting at 1.

You cannot use the syntax for outputting to an ODBC data source to export multiple result sets. Multiple result sets cannot be exported to a Microsoft Excel workbook.

The output format can be specified with the optional FORMAT clause. The default format is TEXT. If no FORMAT clause is specified, then the Interactive SQL output_format option setting is used.

Because the INPUT statement is an Interactive SQL statement, you cannot use it in any compound statement (such as an IF statement), in a stored procedure, or in any statement executed by the server.

Microsoft Excel does not support BINARY or LONG BINARY data, so the data must be converted to a string or number before it is exported.

If the isql_maximum_displayed_rows option is set to a value less than the number of rows in the result set, then at most isql_maximum_displayed_rows are fetched and displayed in the *Results* pane. If an OUTPUT statement is used to log the result set to a file, then the query is re-executed so that the entire result set can be logged. If re-execution is undesirable, then set isql_maximum_displayed_rows to ALL.

Similarly, if the truncation_length option for columns that appear in the *Results* pane is set to a value less than the actual column data width, then the column output is truncated. If an OUTPUT statement is used to log the result set to a file, then the query is re-executed so that the entire result set can be logged. If re-execution is undesirable, then set truncation_length to a sufficiently large value.

Privileges

None.

Side Effects

In Interactive SQL, the *Results* tab displays the results of the current query.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Place the contents of the Employees table in a text file:

```
SELECT * FROM GROUPO.Employees;
```

```
OUTPUT TO 'c:\\temp\\Employees.txt'
FORMAT TEXT;
```

Place the contents of the Employees table at the end of an existing text file, and include any messages about the query in this file as well:

```
SELECT * FROM GROUPO.Employees;
OUTPUT TO 'c:\\temp\\Employees.txt'
APPEND VERBOSE;
```

Suppose you want to export a value that contains an embedded line feed character. A line feed character has the numeric value 10, which you can represent as the string \x0a in a SQL statement. For example, execute the following statement, with HEXADECIMAL set to ON:

```
SELECT CAST ('line1\x0aline2' AS VARBINARY);
OUTPUT TO 'c:\\temp\\file.txt' HEXADECIMAL ON;
```

You get a file with one line in it containing: `0x6c696e65310a6c696e6532`.

If you execute the same statement with HEXADECIMAL set to OFF, then you get: `'line1\x0Aline2'`.

Finally, if you set HEXADECIMAL to ASIS, you get a file with two lines:

```
'line1
line2'
```

You get two lines when you use ASIS because the embedded line feed character has been exported without being converted to a two digit hexadecimal representation, and without being prefixed by anything.

The following example outputs the data from the Customers table to a new table, Customers2:

```
SELECT * FROM Customers;
OUTPUT USING 'DSN=SQL Anywhere 17 Demo;PWD=sql'
INTO "Customers2";
```

The following example copies the Customers table from the sample database to a fictitious database called `mydatabase.db`, using the DRIVER option.

```
SELECT * FROM Customers;
OUTPUT USING 'DRIVER=SQL Anywhere 17;UID=DBA;PWD=passwd;DBF=c:\\test\
\mydatabase.db'
INTO "Customers";
```

The following example copies the Customers table from the SQL Anywhere sample database into a table called Customers in a fictitious UltraLite database, `myULDatabase.db`, using the DRIVER option.

```
SELECT * FROM Customers;
OUTPUT USING 'DRIVER=UltraLite 17;DBF=c:\\test\\myULDatabase.udb'
INTO "Customers";
```

The following example copies the Customers table into a fictitious MySQL database called `mydatabase`, using the DRIVER option.

```
SELECT * FROM GROUPO.Customers;
OUTPUT USING 'DRIVER=MySQL ODBC 5.1
Driver;DATABASE=mydatabase;SERVER=mySQLHost;UID=me;PWD=secret'
INTO "Customers";
```

The following command outputs a file which contains `'one\x0Atwo\x0Athree'`:

```
SELECT 'one\ntwo\nthree';
OUTPUT TO 'c:\\temp\\test.txt' HEXADECIMAL OFF;
```

The following example exports the Customers table into a Microsoft Excel workbook called `customers.xlsb`:

```
SELECT * FROM Customers;
OUTPUT TO 'Customers.xlsb' FORMAT EXCEL
```

Related Information

Data Import and Export
Interactive SQL
Tips on Exporting Data with the OUTPUT Statement
Statements Allowed in Procedures, Triggers, Events, and Batches
Supported Character Sets
Returning Multiple Result Sets
SELECT Statement [page 1362]
INPUT Statement [Interactive SQL] [page 1224]
UNLOAD Statement [page 1453]
output_format Option [Interactive SQL]
default_isql_encoding Option [Interactive SQL]
output_format Option [Interactive SQL]
isql_maximum_displayed_rows Option [Interactive SQL]

1.4.4.211 PARAMETERS Statement [Interactive SQL]

Specifies parameters to an Interactive SQL script file.

Syntax

```
PARAMETERS parameter1, parameter2, ...
```

Remarks

The PARAMETERS statement names the parameters for a script file, so that they can be referenced later in the script file.

Parameters are referenced by putting `{parameter1}` into the file where you want the named parameter to be substituted. There must be no spaces between the braces and the parameter name.

Interactive SQL prompts for missing parameters when it executes a statement that uses the parameter. The presence of a PARAMETERS statement does not in itself cause prompting for missing parameter values.

If the `.SQL` file contains a literal string which happens to contain braces, but which do not enclose a parameter name, Interactive SQL does will not prompt you for a value.

If the PARAMETERS statement lists parameters that are not used in the `.SQL` file, Interactive SQL does not prompt for them and they are not treated as an error.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following Interactive SQL script file takes two parameters.

```
PARAMETERS department_id, file;
SELECT Surname
FROM GROUPO.Employees
WHERE DepartmentID = {department_id}
>#{file}.dat;
```

If you save this script in a file named `test.sql`, you can run it from Interactive SQL using the following command:

```
READ 'test.sql' [100] [data]
```

An output file called `data.dat` is created.

Parameters can appear in literal strings.

```
-- Hello.sql
PARAMETERS yourName;
MESSAGE 'Hello, {yourName}' TO CLIENT;
```

If you save this script in a file named `Hello.sql`, you can run it from Interactive SQL using the following command:

```
READ Hello.sql [World]
```

The message "Hello, World" is displayed in the *History* tab.

The following revised example appears to use two parameters.

```
-- Hello2.sql
PARAMETERS yourName;
MESSAGE 'Hello, {yourName}{end}' TO CLIENT;
```

If you save this script in a file named `Hello2.sql`, you can run it from Interactive SQL using the following command:

```
READ Hello2.sql [World]
```

In this case, `"Hello, World{end}"` is displayed in the *History* tab. Interactive SQL does not prompt you for a parameter called `end` because it was not included in the PARAMETERS list.

The sample database has a Departments table that contains the department ID, the department name, and the employee ID of the person who heads the department. You could write an Interactive SQL script file that updated the name of a department and its head. Here is an example of this script:

```
-- UpdateDepartment.sql
PARAMETERS id, name, head_id;
UPDATE Departments D
    SET D.DepartmentName='{name}', D.DepartmentHeadID={head_id}
    WHERE D.DepartmentID={id};
SELECT * FROM Departments D
    WHERE D.DepartmentID={id};
```

The script is run with a READ statement with three parameters: one for the department ID, one for the new department name, and another for the employee ID of the head of the department. To change the name and head of the R&D department (which has an ID of 100) to Research and headed by employee David Scott, you would run the following statement:

```
READ UpdateDepartment.sql [100] [Research] [501]
```

If the READ statement contains fewer parameters than are declared in a PARAMETERS statement, Interactive SQL prompts for their values when a PARAMETERS statement is encountered in the script file. For example, if you ran the following statement, which omits the new department name and department head, Interactive SQL prompts for the missing values when it executes the PARAMETERS statement:

```
READ UpdateDepartment.sql [100];
```

Related Information

Interactive SQL
READ Statement [Interactive SQL] [page 1317]

1.4.4.212 PASSTHROUGH Statement [SQL Remote]

Starts or stops passthrough mode for SQL Remote administration.

Syntax

Start passthrough for a user ID

```
PASSTHROUGH [ ONLY ] FOR userid, ...
```

Start passthrough all subscribers to a publication

```
PASSTHROUGH [ ONLY ] FOR SUBSCRIPTION
TO [ owner. ]publication-name [ ( constant ) ]
```

Stop passthrough

```
PASSTHROUGH STOP
```

Remarks

In passthrough mode, any SQL statements are executed by the database server, and are also placed into the transaction log to be sent in messages to subscribers. If the ONLY keyword is used to start passthrough mode, the statements are not executed at the server; they are sent to recipients only. When a passthrough session contains calls to stored procedures, the procedures must exist in the server that is issuing the passthrough commands, even if they are not being executed locally at the server. The recipients of the passthrough SQL statements are either a list of user IDs or all subscribers to a given publication. Passthrough mode may be used to apply changes to a remote database from the consolidated database or send statements from a remote database to the consolidated database.

Starting passthrough for subscribers of a publication sends statements to remote databases whose subscriptions are started, and does not send statements to remote databases whose subscriptions are created and not started.

PASSTHROUGH STOP stops passthrough mode on the current connection. You must execute the PASSTHROUGH STOP statement on the same connection that initiated the passthrough mode. If start passthrough mode on a connection and it disconnects before a PASSTHROUGH STOP statement is executed, the disconnect implicitly executes a PASSTHROUGH STOP statement.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

```
PASSTHROUGH FOR Sam_Singer ;
...
( SQL statements to be executed at the remote database )
...
PASSTHROUGH STOP ;
```

Related Information

SQL Remote Passthrough Mode
Start and Stop Passthrough Mode
Passthrough Mode Limitations
Upgrades and Resynchronization
Subscription Resynchronization
FORWARD TO Statement [page 1170]
UNLOAD Statement [page 1453]
Troubleshooting: Connectivity Tests for Remote Data Access
Changes to Avoid on a Running System

1.4.4.213 PIVOT Clause

Pivots a table expression in the FROM clause of a SELECT statement (FROM `pivoted-derived-table`) into a pivoted derived table. Pivoted derived tables offer an easy way to rotate row values from a column in a table expression into multiple columns, and perform aggregation where needed on the columns included in the result set.

Syntax

```
FROM pivoted-derived-table
```

```
pivoted-derived-table :
pivot-source-table PIVOT [ XML ] ( pivot-clause ) [ AS ] pivoted-correlation-
name

pivot-source-table : table-expression

pivot-clause :
   aggregate-clause pivot-for-clause pivot-in-clause

aggregate-clause :
aggregate-function ( [ aggregate-expression ] ) [ [ AS ] aggregate-alias ]
[,...]

pivot-for-clause :
FOR pivot-column
| FOR ( pivot-column [,...] )

pivot-in-clause :
IN ( constant-expression [ [ AS ] constant-expression-alias ] [,...] )
| IN ( ( constant-expression [,...] ) [ [ AS ] constant-expression-alias ]
[,...] )
| IN variable-name
| IN ( subquery-expression )
| IN ( ALL )
| IN ( ANY )
```

Parameters

XML clause

Specify XML to output the aggregates and pivot values in a new, single column in XML format. Use this clause when specifying the values of `pivot-in-clause` using a non-constant list of values. The XML clause is used in conjunction with the IN (ANY), IN (ALL), and IN (`subquery-expression`) clauses.

aggregate-function ([aggregate-expression]) [[AS] aggregate-alias]

Specify any aggregate function that returns a single value for a set of rows.

- `aggregate-function` – specify any aggregate function that returns a single value for a set of rows (also known as a scalar function).
- `aggregate-expression` – specify the parameters of the aggregate function. The aggregate expression must reference only columns found in `pivot-source-table`.
  If the aggregate function allows it, then the ORDER BY and DISTINCT clauses can be specified. For example:

  ```
  LIST( DISTINCT DepartmentID ORDER BY Salary )
  ```

- `aggregate-alias` – specify an alias for the aggregate function. The list of aggregate functions can have at most one aggregate function without an alias. Aggregate aliases, together with the IN clause aliases, are used to generate the names of the new columns of the pivoted derived table. As a best practice, always specify an alias for your aggregate.

FOR clause

Specify one or more columns on which to pivot the data. `pivot-column` must be a column in `pivot-source-table`. If you specify more than one `pivot-column`, then you must enclose them in parentheses.

IN (constant-expression [[AS] constant-expression-alias] [,...])

Specify a set of constant expressions on which to pivot the data. Use this syntax when the FOR clause lists only one column, namely, FOR `pivot-column`.

If an alias is not specified, then the implicit alias is the string representing the constant expression. For example, the implicit alias for the constant 10 is "10." Always specify an alias for `constant-expression`. Use implicit and explicit aliases for constant expressions in the IN clause, together with aggregate aliases, to generate the names of the new columns of the pivoted derived table.

Each new column in the pivoted derived table corresponds to a pairing of an aggregate function and an IN item, and has a name that reflects the pairing. The first part of the name is the alias of the IN item, and the second part of the name (after the underscore) is the alias of the aggregate function. If a generated column name is an invalid identifier, then an error is generated and the statement fails.

IN ((constant-expression [,...]) [[AS] constant-expression-alias] [,...])

Specify a set of lists containing constant expressions on which to pivot the data. Use this form of the IN clause when you specify multiple columns in the FOR clause. The number of columns specified in the FOR clause must be equal to the number of items in any constant list of the IN clause.

If an alias is not specified, then the implicit alias is the string representing the list of constant expressions. For example, the implicit alias for the constant list (10, 20) is "(10, 20)". Implicit and explicit aliases for constant expressions in the IN clause, together with the aggregate aliases, are used to generate the names of the new columns of the pivoted derived table.

IN variable-name

Specify a variable that contains a set of constants on which to pivot the data. This form of IN clause is similar to the IN clause using constants. All the conditions described above for the IN clause with constants must hold for the IN clause using a variable.

The variable must be set when the statement containing the pivoted derived table is described, open, or run. The variable must be declared using an array type. The aliases for the IN clause are implicitly defined using the current values of the variable. For example, if the variable is the array (10, 20), the alias for first constant is "10", and the alias for the second constant is "20".

If the FOR clause has only one pivot column, the variable must be an array with elements that have a domain that is compatible with the data in `pivot-column`. For example, if the FOR clause is `FOR DepartmentID`, with a DepartmentID column of type INT, then a variable @var defined as an array of type INT can be used in the IN clause.

If the FOR clause has a set of pivot columns, then the variable must be an array of rows whose elements have compatible domains with the columns in the FOR clause. For example, if the FOR clause is `FOR (DepartmentID, State)`, with a DepartmentID column of type INT, and State column of type CHAR(16), then a variable @var defined as an array of type ROW(X INT, Y CHAR(16)) can be used in the IN clause.

IN (subquery-expression)

Specify `IN ( subquery-expression )` to pivot on all values found by the subquery.

The XML column in the derived table has a name defined by the names of the columns in the FOR clause. It is an XML element containing a set of items, one item for each pair of values for pivot columns and aggregate aliases. The NULL values are encoded as empty string in the XML element.

If the FOR clause is FOR pivot-column (for example, COUNT(*) AS "COUNT" FOR DepartmentID), then the format of each item in the XML column is:

```
<item>
<column name="DepartmentID">100</column>
<column name="COUNT">1</column>
</item>
```

Specify the XML keyword when using this form of the IN clause.

IN (ALL)

Specify IN (ALL) to pivot all values in pivot-source-table.

IN (ALL) is equivalent to IN (subquery-expression), where subquery-expression is pivot-source-table.

Specify the XML keyword when using this form of the IN clause.

IN (ANY)

Specify IN (ANY) to pivot on values in pivot-source-table. If ANY is specified, then the XML column is similar to the XML column generated for IN (ALL) but the NULL values for the aggregates are eliminated from the XML string. This form of the IN clause results in a more compact XML column.

A pivoted derived table with an IN (ANY) clause is similar to a derived table specified with IN(ALL) clause with the exception that NULL values are not included in the XML string of the extra column.

Specify the XML keyword when using this form of the IN clause.

Remarks

The definition of a pivoted derived table contains an input table expression, pivot-source-table. The columns and values to pivot on are defined in the FOR and IN clauses. The grouping columns of the pivoted derived tables are a subset of the columns of pivot-source-table. A pivoted derived table is computed by grouping pivot-source-table on the grouping columns and then computing the aggregate functions specified in the aggregate clause. The pivoted derived table has a column for each value of the grouping columns. There are extra columns added to the pivoted derived table, one for each pair of an item in the aggregate clause and an item in the IN clause. The values of the new columns are the aggregate functions specified in the aggregate clause. The names of these new columns are generated from the aliases specified in the aggregate clause for aggregate functions, and the aliases and values specified in the IN clause. In total, if A aggregate functions are specified, and the IN clause has I elements, then there are A x I extra columns. However, if the XML clause is specified, then only one extra column is added. This column contains the aggregates, in the XML format, for all of the combinations of the values in the IN clause and the aggregate clause, in one string.

Privileges

You must have SELECT privileges on the objects referenced in pivot-source-table.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example selects data from the Employees table and pivots it on the DepartmentID column, where the Department ID is 100, 200, 300, 400, or 500.

```
SELECT *
FROM ( SELECT DepartmentID, State, Salary
       FROM   Employees
       WHERE State IN ( 'OR', 'CA', 'AZ', 'UT' )
     ) MyPivotSourceData
   PIVOT (
      SUM( Salary ) TotalSalary
      FOR DepartmentID IN ( 100, 200, 300, 400, 500 )
   ) MyPivotedData
ORDER BY State;
```

STATE	100_TotalSalary	200_TotalSalary	300_TotalSalary	400_TotalSalary	500_TotalSalary
AZ	(NULL)	(NULL)	93,732.000	(NULL)	85,300.800
CA	(NULL)	156,600.000	(NULL)	(NULL)	(NULL)
OR	(NULL)	47,653.000	(NULL)	80,339.000	54,790.000
UT	306,318.690	37,900.000	31,200.000	107,129.000	59,479.000

In the results, the aggregate alias and the values for DepartmentID are included in the column names of the result set (for example, 100_TotalSalary) to clarify which value is being pivoted. The column names in this example mean "the total salary for department X". The salaries for employees in each State/DepartmentID tuple are aggregated (in this case, summed together).

The following is an example of the IN `variable-name` syntax when you have only one pivot column. The variable must be an array with elements of compatible domains with `pivot-column`:

```
CREATE VARIABLE @var INT ARRAY;
SET @var = ( SELECT ARRAY_AGG( DISTINCT  DepartmentID ORDER BY Salary ) FROM
Employees )
SELECT *
FROM ( SELECT DepartmentID,State, Salary FROM Employees ) p
PIVOT  (
        SUM(Salary) S, COUNT(*) C
        FOR DepartmentID in @var
```

```
) PivotTable
```

```
CREATE VARIABLE @var INT ARRAY;
SET @var = ( SELECT ARRAY_AGG( DISTINCT  DepartmentID ORDER BY Salary ) FROM
Employees )
SELECT *
FROM ( SELECT DepartmentID,State, Salary FROM Employees ) p
PIVOT  (
        SUM(Salary) S, COUNT(*) C
        FOR DepartmentID in @var
) PivotTable
```

The following is an example of the IN `variable-name` syntax when you have a set of pivot columns. The variable must be an array with rows whose elements have compatible domains with the columns in the FOR clause:

```
CREATE VARIABLE @var ROW( DeptID INT, St CHAR(16)) ARRAY;
SET @var = ( SELECT ARRAY_AGG( DISTINCT ROW( DepartmentID, State) ORDER BY
Salary ) FROM Employees )
SELECT *
FROM ( SELECT DepartmentID,State, Salary FROM Employees ) p
PIVOT (
    SUM ( Salary ) S, COUNT(*) C
    FOR ( DepartmentID, State ) IN @var
) PivotTable
```

The following statements compute the same pivoted derived table using the different supported PIVOT constructs. Different PIVOT constructs offer alternative ways to generated desired pivoted derived tables where the IN clause is dynamically generated based on the current data.

PIVOT (... IN (constant-list))

You can use this form when you know what constants to be specify in the IN clause.

```
SELECT *
   FROM ( SELECT DepartmentID, City FROM Employees) E
   PIVOT  (
     COUNT(*) AS C
     FOR DepartmentID IN ( 100, 200, 300 ) ) AS PivotedTable
 ORDER BY City;
```

City	100_C	200_C	300_C
Charlottetown	0	2	0
Cornwall	2	1	1
Elora	0	1	0
...	...	...	...

PIVOT (... IN (generated-constant-list))

You can use a dynamically generated IN clause to generate the desired IN clause based on the current data in the Employees table:

```
BEGIN
DECLARE stmt LONG VARCHAR;
 SET stmt =    ( SELECT 'SELECT * INTO dba.ConstantsTable
        FROM ( SELECT DepartmentID, City FROM Employees ) p
        PIVOT  (
```

```
            COUNT(*) AS C
            FOR DepartmentID IN ( ' + ( SELECT LIST( DepartmentID )
     FROM ( SELECT DISTINCT DepartmentID FROM Employees WHERE DepartmentID <
400 ) T )
   +  ' ) )  AS PivotedTable
 ORDER BY City' );
EXECUTE IMMEDIATE stmt;
END;
```

After the statements finish, execute the following statement to view the content of the pivoted derived table, XMLTable, that you created:

```
SELECT * FROM XMLTable;
```

City	100_C	200_C	300_C
Charlottetown	0	2	0
Cornwall	2	1	1
Elora	0	1	0
...	...	...	...

PIVOT (... IN variable-name)

The IN clause can be specified using a variable of type ARRAY which can be set to the desired set of values.

```
BEGIN
DROP TABLE IF EXISTS VarTable;
DECLARE @var INT ARRAY;
SET @var = ( SELECT  ARRAY_AGG( DepartmentID )  FROM  ( SELECT DISTINCT
DepartmentID FROM Employees WHERE DepartmentID < 400 ) T ) ;
SELECT * INTO dba.VarTable
        FROM (SELECT DepartmentID, City FROM Employees) E
        PIVOT  (
            COUNT(*) AS C
            FOR DepartmentID IN @var ) AS PivotedTable
ORDER BY City;
END
SELECT * from VarTable;
```

City	100_C	200_C	300_C
Charlottetown	0	2	0
Cornwall	2	1	1
Elora	0	1	0
...	...	...	...

PIVOT XML(... IN (subquery-expression))

The following example generates a pivoted derived table in XML format, and then extracts the pivoted derived table from the data. You can use this form when the constants in the IN clause are not known.

```
BEGIN
DECLARE qry LONG VARCHAR;
DECLARE city_c INT;
DECLARE total_c INT;
CREATE OR REPLACE VARIABLE globalvals ARRAY OF ROW( City CHAR(26),
DepartmentID INT, C INT );
```

```
SELECT COUNT(*) total_c, COUNT( DISTINCT city ) city_c,
ARRAY_AGG( ROW ( City, DepartmentID, C ) ORDER BY city, DepartmentID )  INTO
total_c, city_c, globalvals
FROM (
SELECT PivotedTable.City, DepartmentID, C
FROM   ( SELECT *
        FROM (SELECT DepartmentID, City FROM Employees) E
        PIVOT  XML (
            COUNT(*) AS C
            FOR DepartmentID IN ( SELECT DepartmentID FROM Employees WHERE
DepartmentID < 400 ) )  AS EEE  ) AS PivotedTable,
LATERAL (SELECT * FROM  openxml( PivotedTable.[DepartmentID_xml], '/PivotSet/
item')
         WITH ( DepartmentID INT 'column[@name="DepartmentID"]',  C  INT
'column[@name="C"]' ) ) XXX
) AS  dt;
SELECT
 'SELECT ( globalvals[[ row_num ]]).city AS City,
 ' || LIST('(globalvals[[row_num+ ' || row_num || ']]).C AS ' || STRING( '[',
globalvals[[row_num+1]].DepartmentID, '_C]') ) ||  '
 INTO dba.XMLTable FROM sa_rowgenerator(1, ' || total_c || ' , ' || (total_c/
city_c) || ' )' INTO qry
FROM sa_rowgenerator( 0, (total_c/city_c) -1);
DROP TABLE IF EXISTS XMLTable;
EXECUTE IMMEDIATE qry;
DROP VARIABLE IF EXISTS globalvals;
END;
```

After the example finishes, execute the following statement to view the content of the pivoted derived table, XMLTable, that you created:

```
SELECT * FROM XMLTable;
```

City	100_C	200_C	300_C
Charlottetown	0	2	0
Cornwall	2	1	1
Elora	0	1	0
...	...	...	...

Related Information

Tutorial: Pivoting Table Data
FROM Clause [page 1173]
UNPIVOT Clause [page 1459]

1.4.4.214 PREPARE Statement [ESQL]

Prepares a statement to be executed later, or defines a cursor.

Syntax

```
PREPARE statement-name
   FROM statement [ FOR { UPDATE [ cursor-concurrency ] | READ ONLY } ]
[ DESCRIBE describe-type INTO [ [ SQL ] DESCRIPTOR ] descriptor ]
[ WITH EXECUTE ]
```

```
statement-name :  identifier | hostvar
```

```
statement :   string | hostvar
```

```
describe-type :
  [ ALL | BIND VARIABLES | INPUT | OUTPUT | SELECT LIST ]
  [ LONG NAMES [ [ [ OWNER. ]TABLE. ]COLUMN ]
    | WITH VARIABLE RESULT ]
```

```
cursor-concurrency :
BY { VALUES | TIMESTAMP | LOCK }
```

Parameters

statement-name

The statement name can be an identifier or host variable. However, do not use an identifier when using multiple SQLCAs. If you do, two prepared statements may have the same statement number, which could cause the wrong statement to be executed or opened. Also, using an identifier for a statement name is not recommended for multithreaded applications where the statement name may be referenced by multiple threads concurrently.

DESCRIBE clause

If DESCRIBE INTO DESCRIPTOR is used, the prepared statement is described into the specified descriptor. The describe type may be any of the describe types allowed in the DESCRIBE statement.

FOR UPDATE | FOR READ ONLY

Defines the cursor updatability if the statement is used by a cursor. A FOR READ ONLY cursor cannot be used in an UPDATE (positioned) or a DELETE (positioned) operation. FOR READ ONLY is the default.

In response to any request for a cursor that specifies FOR UPDATE, the database server provides either a value-sensitive cursor or a sensitive cursor. Insensitive and asensitive cursors are not updatable.

BY VALUES | BY TIMESTAMP

The database server uses a keyset-driven cursor to enable the application to be informed when rows have been modified or deleted as the result set is scrolled.

BY LOCK clause

The database server acquires intent row locks on fetched rows of the result set. These are long-term locks that are held until the transaction is committed or rolled back.

WITH EXECUTE clause

If the WITH EXECUTE clause is used, the statement is executed if and only if it is not a CALL or SELECT statement, and it has no host variables. The statement is immediately dropped after a successful execution. If the PREPARE and the DESCRIBE (if any) are successful but the statement cannot be executed, a warning SQLCODE 111, SQLSTATE 01W08 is set, and the statement is not dropped.

The DESCRIBE INTO DESCRIPTOR and WITH EXECUTE clauses may improve performance because they cut down on the required client/server communication.

WITH VARIABLE RESULT clause

The WITH VARIABLE RESULT clause is used to describe procedures that may have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database server sets the SQLCOUNT value after the describe to one of the following values:

0

The result set may change: the procedure call should be described again following each OPEN statement.

1

The result set is fixed. No re-describing is required.

> **i** Note
>
> For compatibility reasons, preparing COMMIT, PREPARE TO COMMIT, and ROLLBACK statements is still supported. However, you should perform all transaction management operations with static Embedded SQL because certain application environments may require it. Also, other Embedded SQL systems do not support dynamic transaction management operations.

Remarks

The PREPARE statement prepares a SQL statement from the `statement` and associates the prepared statement with `statement-name`. This statement name is referenced to execute the statement, or to open a cursor if the statement is a SELECT or CALL statement. The `statement-name` may be a host variable of type a_sql_statement_number defined in the `sqlca.h` header file that is automatically included. If an identifier is used for the `statement-name`, only one statement per module may be prepared with this `statement-name`.

If a host variable is used for `statement-name`, it must have the type short int. There is a typedef for this type in `sqlca.h` called a_sql_statement_number. This type is recognized by the SQL preprocessor and can be used in a DECLARE section. The host variable is defined by the database during the PREPARE statement, and you do not need to initialize it.

Privileges

None.

Side Effects

Any statement previously prepared with the same name is lost.

The statement is dropped after use only if you use WITH EXECUTE and the execution is successful. Ensure that you DROP the statement after use in other circumstances. If you do not, the memory associated with the statement is not reclaimed.

Standards

ANSI/ISO SQL Standard

PREPARE is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The optional FOR UPDATE, FOR READ ONLY, DESCRIBE, and WITH EXECUTE clauses are not in the standard.

Example

The following statement prepares a simple query:

```
EXEC SQL PREPARE employee_statement FROM
'SELECT Surname FROM Employees';
```

Related Information

Dynamic SQL Statements
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
DESCRIBE Statement [ESQL] [page 1076]
OPEN Statement [ESQL] [SP] [page 1286]
EXECUTE Statement [ESQL] [page 1151]
DROP STATEMENT Statement [ESQL] [page 1125]
Statement cannot be executed

1.4.4.215 PREPARE TO COMMIT Statement

Checks whether a COMMIT can be performed successfully.

Syntax

PREPARE TO COMMIT

Remarks

The PREPARE TO COMMIT statement tests whether a COMMIT can be performed successfully. The statement will cause an error if a COMMIT is impossible without violating the integrity of the database.

The PREPARE TO COMMIT statement cannot be used in stored procedures, triggers, events, or batches.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following sequence of statements leads to an error because of foreign key checking on the Employees table.

```
EXECUTE IMMEDIATE
   "SET OPTION wait_for_commit = 'On'";
EXECUTE IMMEDIATE "DELETE FROM Employees
   WHERE EmployeeID = 160";
EXECUTE IMMEDIATE "PREPARE TO COMMIT";
```

The following sequence of statements does not cause an error when the delete statement is executed, even though it causes integrity violations. The PREPARE TO COMMIT statement returns an error.

```
SET OPTION wait_for_commit= 'On';
DELETE
FROM GROUPO.Departments
WHERE DepartmentID = 100;
PREPARE TO COMMIT;
```

Related Information

COMMIT Statement [page 811]
ROLLBACK Statement [page 1355]

1.4.4.216 PRINT Statement [T-SQL]

Returns a message to the client, or display a message in the database server messages window.

Syntax

```
PRINT format-string [, arg-list ]
```

Remarks

The PRINT statement returns a message to the client window if you are connected from an Open Client application or jConnect application. If you are connected from an Embedded SQL or ODBC application, the message is displayed in the database server messages window.

The format string can contain place holders for the arguments in the optional argument list. These place holders are of the form `%nn!`, where `nn` is an integer between 1 and 20.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement displays a message:

```
PRINT 'Display this message';
```

The following statement illustrates the use of placeholders in the PRINT statement:

```
DECLARE @var1 INT, @var2 INT
SELECT @var1 = 3, @var2 = 5
PRINT 'Variable 1 = %1!, Variable 2 = %2!', @var1, @var2
```

Related Information

MESSAGE Statement [page 1278]

1.4.4.217 PUT Statement [ESQL]

Inserts a row into the specified cursor.

Syntax

```
PUT cursor-name
{ USING DESCRIPTOR sqlda-name | FROM hostvar-list }
[ INTO { DESCRIPTOR sqlda-name | hostvar-list } ]
[ ARRAY :row-count ]
```

```
cursor-name : identifier | hostvar
```

```
sqlda-name : identifier
```

```
hostvar-list : may contain indicator variables
```

```
row-count : integer | hostvar
```

Remarks

Inserts a row into the named cursor. Values for the columns are taken from the first SQLDA or the host variable list, in a one-to-one correspondence with the columns in the INSERT statement (for an INSERT cursor) or the columns in the SELECT list (for a SELECT cursor).

The PUT statement can be used only on a cursor over an INSERT or SELECT statement that references a single table in the FROM clause, or that references an updatable view consisting of a single base table.

If the sqldata pointer in the SQLDA is the null pointer, no value is specified for that column. If the column has a DEFAULT VALUE associated with it, that is used; otherwise, a NULL value is used.

The second SQLDA or host variable list contains the results of the PUT statement.

The optional ARRAY clause can be used to perform wide puts, which insert more than one row at a time and which may improve performance. The integer value is the number of rows to be inserted. The SQLDA must contain a variable for each entry (number of rows * number of columns). The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.

> **i** Note
>
> For scroll (value-sensitive) cursors, the inserted row will appear if the new row matches the WHERE clause and the keyset cursor has not finished populating. For dynamic cursors, if the inserted row matches the WHERE clause, the row may appear. Insensitive cursors cannot be updated.

Privileges

You must be the owner of the tables referenced in the cursor, or have INSERT privileges on the tables, or have the INSERT ANY TABLE system privilege.

Side Effects

When inserting rows into a value-sensitive (keyset driven) cursor, the inserted rows appear at the end of the result set, even when they do not match the WHERE clause of the query or if an ORDER BY clause would normally have placed them at another location in the result set.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement illustrates the use of PUT in Embedded SQL:

```
EXEC SQL PUT cur_employee FROM :employeeID, :surname;
```

Related Information

Cursors Used to Modify Rows
UPDATE Statement [page 1463]
UPDATE (Positioned) Statement [ESQL] [SP] [page 1470]
DELETE Statement [page 1070]
DELETE Statement (Positioned) [ESQL] [SP] [page 1074]
INSERT Statement [page 1232]
SET Statement [page 1373]

1.4.4.218 RAISERROR Statement

Signals an error and sends a message to the client.

Syntax

```
RAISERROR error-number [ format-string ] [, arg-list ]
```

Parameters

error-number

The `error-number` is a five-digit integer greater than 17000. The error number is stored in the global variable @@error.

format-string

`format-string` is string up to 255 bytes in length. It can contain placeholders for the arguments in the optional argument list. These placeholders are of the form %`nn`!, where `nn` is an integer between 1 and 20.

Remarks

The RAISERROR statement allows user-defined errors to be signaled and sends a message on the client.

To create new error messages, use the CREATE ERROR statement. To view the messages in the ISYSUSERMESSAGE system table, query the SYSUSERMESSAGE system view.

The extended values supported by the Adaptive Server Enterprise RAISERROR statement are not supported in SQL Anywhere.

If `format-string` is not supplied or is empty, the error number is used to locate an error message in the system tables. Adaptive Server Enterprise obtains messages 17000-19999 from the SYSMESSAGES table. In SQL Anywhere this table is an empty view, so errors in this range should provide a format string. Messages for error numbers of 20000 or greater are obtained from the ISYSUSERMESSAGE system table.

Intermediate RAISERROR status and code information is lost after the procedure terminates. If at return time an error occurs along with the RAISERROR then the error information is returned and the RAISERROR information is lost. The application can query intermediate RAISERROR statuses by examining @@error global variable at different execution points.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example uses RAISERROR to disallow connections.

```
CREATE PROCEDURE DBA.login_check()
BEGIN
    // Allow a maximum of 3 concurrent connections
    IF( DB_PROPERTY('ConnCount') > 3 ) THEN
         RAISERROR 28000
      'User %1! is not allowed to connect -- there are ' ||
                    'already %2! users logged on',
      Current User,
      CAST( DB_PROPERTY( 'ConnCount' ) AS INT )-1;
    ELSE
         CALL sp_login_environment;
    END IF;
END
go
GRANT EXECUTE ON DBA.login_check TO PUBLIC
go
SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

Related Information

BEGIN Statement [page 784]
CREATE MESSAGE Statement [T-SQL] [page 899]
SYSUSERMESSAGE System View [page 1968]
CREATE TRIGGER Statement [T-SQL] [page 1043]
CREATE TRIGGER Statement [page 1036]
on_tsql_error Option
login_procedure Option
continue_after_raiserror Option
SIGNAL Statement [SP] [page 1405]

1.4.4.219 READ Statement [Interactive SQL]

Reads Interactive SQL statements from a file.

Syntax

```
READ [ ENCODING encoding ] filename [ parameter ] ...
```

```
encoding : identifier | string
```

Parameters

ENCODING

The ENCODING clause allows you to specify the encoding that is used to read the file. The READ statement does not process escape characters when it reads a file. It assumes that the entire file is in the specified encoding.

When running Interactive SQL, the encoding that is used to read the data is determined in the following order:

1. The encoding specified by the ENCODING clause (if this clause is specified).
2. The encoding specified by the byte order mark (BOM) in the file (if a BOM is specified).
3. The encoding specified with the default_isql_encoding option (if this option is set).
4. The default encoding for the platform you are running on. On English Windows computers, the default encoding is 1252.

Remarks

The READ statement reads a sequence of Interactive SQL statements from the named file. This file can contain any valid Interactive SQL statements, including other READ statements. READ statements can be nested to any depth.

If `filename` has no file extension, Interactive SQL searches for the same file name with the extension `.sql`.

If `filename` does not contain an absolute path, Interactive SQL searches for the file. The location of `filename` is determined based on the location of the READ statement, as follows:

- If the READ statement is executed directly in Interactive SQL, Interactive SQL first attempts to resolve the path to `filename` relative to the directory in which Interactive SQL is running. If unsuccessful, Interactive SQL looks for `filename` in the directories specified in the environment variable SQLPATH, and then the directories specified in the environment variable PATH.
- If the READ statements reside in an external file (for example, a `.sql` file), Interactive SQL first attempts to resolve the path to `filename` relative to the location of the external file. If unsuccessful, Interactive SQL looks for `filename` in a path relative to the directory in which Interactive SQL is running. If still unsuccessful, Interactive SQL looks in the directories specified in the environment variable SQLPATH, and then the directories specified in the environment variable PATH.

Parameters can be listed after the name of the SQL script file. These parameters correspond to the parameters named in the PARAMETERS statement at the beginning of the statement file.

Parameter names must be enclosed in square brackets. Interactive SQL substitutes the corresponding parameter wherever the source file contains `{ parameter-name }`, where `parameter-name` is the name of the appropriate parameter.

The parameters passed to a script file can be identifiers, numbers, quoted identifiers, or strings. When quotes are used around a parameter, the quotes are put into the text during the substitution. Parameters that are not identifiers, numbers, or strings (contain spaces or tabs) must be enclosed in square brackets ([ ]). This allows for arbitrary textual substitution in the script file.

If not enough parameters are passed to the script file, Interactive SQL prompts for values for the missing parameters.

When executing a `reload.sql` file with Interactive SQL, you must specify the encryption key as a parameter. If you do not provide the key in the READ statement, Interactive SQL prompts for the key.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Suppose you have a script file, `myscript.sql`, which contains the following SQL statements:

```
PARAMETERS par1, par2;
BEGIN
DECLARE v_par1 int;
DECLARE v_par2 varchar(200);
SET v_par1 = {par1};
SET v_par2 = {par2};
MESSAGE STRING('PAR1 Value: ', v_par1 ) TO CLIENT;
MESSAGE STRING('PAR2 Value: ', v_par2 ) TO CLIENT;
END;
```

Executing the following READ statement returns executes the commands in `c:\temp\myscript.sql`, and replaces par1 with value 123, and par2 with value 041028.

```
READ 'c:\\temp\\myscript.sql' 123 '041028';
```

The READ statement returns the following result:

```
PAR1 Value: 123
PAR2 Value: 041028
```

The following example starts Interactive SQL and instructs it to process a fictitious file that uses a specific OEM codepage:

```
dbisql READ ENCODING 'cp437' myfile.sql
```

Related Information

Interactive SQL
PARAMETERS Statement [Interactive SQL] [page 1296]
default_isql_encoding Option [Interactive SQL]
Interactive SQL Utility (dbisql)

1.4.4.220 READTEXT Statement [T-SQL]

Reads text and image values from the database, starting from a specified offset and reading a specified number of bytes. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

Syntax

```
READTEXT table-name.column-name
text-pointer-offset text-size
[ HOLDLOCK ]
```

Remarks

READTEXT is used to read CHAR, NCHAR, and BINARY columns from a database. You cannot perform READTEXT operations on views.

Privileges

You must be the owner or have SELECT privileges on the table, or have the SELECT ANY TABLE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following Embedded SQL example uses TEXTPTR to locate the Description column associated with ProductID 500 in the MarketingInformation table.

The text pointer is stored in the variable txtptr and supplied as a parameter to the READTEXT statement which returns 55 bytes, starting at column offset 181.

```
EXEC SQL BEGIN DECLARE SECTION;
char            hostvar[100];
EXEC SQL END DECLARE SECTION;
EXEC SQL create variable txtptr binary(16);
EXEC SQL set txtptr =
    ( SELECT textptr(Description)
        FROM GROUPO.MarketingInformation
        WHERE ProductID = '500' );
EXEC SQL PREPARE S1 FROM
    'READTEXT GROUPO.MarketingInformation.Description txtptr 181 55';
EXEC SQL EXECUTE S1 INTO :hostvar;
printf( "hostvar: %s\n", hostvar );
```

READTEXT returns the following string.

```
Lightweight 100% organically grown cotton construction.
```

Related Information

WRITETEXT Statement [T-SQL] [page 1491]
GET DATA Statement [ESQL] [page 1185]
TEXTPTR Function [Text and Image] [page 588]

1.4.4.221 REFRESH MATERIALIZED VIEW Statement

Initializes or refreshes the data in a materialized view by executing its query definition.

Syntax

```
REFRESH MATERIALIZED VIEW view-list
  [ WITH {
        ISOLATION LEVEL isolation-level
      | { EXCLUSIVE | SHARE } MODE } ]
  [ FORCE BUILD ]
```

```
view-list :
[ owner.]materialized-view-name [, ... ]
```

```
isolation-level :
READ UNCOMMITTED
| READ COMMITTED
| SERIALIZABLE
| REPEATABLE READ
| SNAPSHOT
```

Parameters

WITH clause

Use the WITH clause to specify the type of locking to use on the underlying base tables during the refresh. The type of locking determines how the materialized view is populated and how concurrency for transactions is affected. The WITH clause setting does not impact the type of lock placed on the materialized view itself, which is always an exclusive lock. The possible locking clauses you can specify are:

ISOLATION LEVEL isolation-level

Use WITH ISOLATION LEVEL to change the isolation level for the execution of the refresh operation. The original isolation level is restored for the connection when the statement completes.

For immediate views, `isolation-level` can only be SERIALIZABLE.

For snapshot isolation, only the transaction snapshot level is supported by the REFRESH MATERIALIZED VIEW statement. Specify SNAPSHOT as the isolation level. The statement-snapshot and readonly-statement-snapshot levels are not supported.

EXCLUSIVE MODE

Use WITH EXCLUSIVE MODE if you do not want to change the isolation level, but want to guarantee that the data is updated to be consistent with committed data in the underlying tables. When using WITH EXCLUSIVE MODE, exclusive table locks are placed on all underlying base tables and no other transaction can execute queries, updates, or any other action against the underlying table(s) until the refresh operation is complete. If exclusive table locks cannot be obtained, the refresh operation fails and an error is returned.

SHARE MODE

Use WITH SHARE MODE to give read access on underlying tables to other transactions while the refresh operation takes place. When this clause is specified, shared table locks are obtained on all underlying base tables before the refresh operation is performed and until the refresh operation completes.

FORCE BUILD clause

By default, when you execute a REFRESH MATERIALIZED VIEW statement, the database server checks whether the materialized view is stale (that is, underlying tables have changed since the materialized view was last refreshed). If it is not stale, the refresh does not take place. Specify the FORCE BUILD clause to force a refresh of the materialized view regardless of whether the materialized view is stale.

Remarks

Use this statement to initialize or refresh the materialized views listed in `view-list`.

If a REFRESH MATERIALIZED VIEW statement is executed against a materialized view that is not stale, a refresh is not performed unless the FORCE BUILD clause is specified.

The default refresh behavior for locking and data concurrency is as follows:

- If the view is an immediate view, the default refresh behavior is WITH SHARE MODE, regardless of whether snapshot isolation is enabled.

- If the view is a manual view and snapshot isolation *is in use*, the default is WITH ISOLATION LEVEL SNAPSHOT.
- If the view is a manual view and snapshot isolation *is not in use*, the default is WITH SHARE MODE.

Several options must have specific values for a REFRESH MATERIALIZED VIEW to succeed, and for the view to be used in optimization. Additionally, there are option settings that are stored for each materialized view when it is created. To refresh the view, or to use the view in optimization these option settings must match the current options.

When a refresh fails after having done partial work, the view is left in an uninitialized state, and the data cannot be restored to what it was before the refresh started. Examine the error that occurred when the refresh failed, resolve the issue that caused the failure, and execute the REFRESH MATERIALIZED VIEW statement again.

You can also use the IMMEDIATE REFRESH clause of the ALTER MATERIALIZED VIEW statement to change the view to be refreshed immediately when underlying data changes.

This statement cannot be executed when the connection has cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

You must be the owner of the materialized view or have INSERT privilege on it. Additionally, you must be the owner of the underlying tables, or have SELECT privilege on them, or have the SELECT ANY TABLE system privilege.

Side Effects

Any open cursors that reference the materialized view are closed.

A checkpoint is performed at the beginning of execution.

Automatic commits are performed at the beginning and end of execution.

While executing, an exclusive schema lock is placed on the materialized view being refreshed using the connection blocking option, and shared schema locks, without blocking, are placed on all tables referenced by the materialized view. If the WITH clause is specified, extra locks may be acquired on the underlying tables. Also, until refreshing is complete, the materialized view is in an uninitialized state, making it unavailable to the database server for either query optimization or query execution.

If the REFRESH MATERIALIZED VIEW statement executes using snapshot isolation, the database's transaction log will contain both the REFRESH statement text and copies of all of the individual rows that are inserted to the materialized view. The individual rows are necessary to ensure that, should the database require recovery, the contents of the view after recovery matches precisely the view's contents upon the original completion of the REFRESH MATERIALIZED VIEW statement. Moreover, the individual rows in the transaction log are applied individually when the database is mirrored. For this reason, you may want to limit the frequency of REFRESH MATERIALIZED VIEW statements when using snapshot isolation, or truncate the transaction log periodically, using the BACKUP DATABASE statement, to reduce the amount of disk space required for the transaction log.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Suppose you create a materialized view, EmployeeConfid99, and then populate it with data using the following statements:

```
CREATE MATERIALIZED VIEW EmployeeConfid99 AS
   SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary,
ManagerID,
      Departments.DepartmentName, Departments.DepartmentHeadID
   FROM GROUPO.Employees, GROUPO.Departments
   WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid99;
```

Later, after the view has been in use, you want to refresh the view using the READ COMMITTED isolation level (isolation level 1), and you want the view to be rebuilt. You could execute the following statement:

```
REFRESH MATERIALIZED VIEW EmployeeConfid99
   WITH ISOLATION LEVEL READ COMMITTED
   FORCE BUILD;
```

> ⚠ Caution
>
> When you are done with this example, drop the materialized view you created (`DROP MATERIALIZED VIEW EmployeeConfid99`). Otherwise, you cannot make schema changes to its underlying tables, Employees and Departments, when trying out other examples. You cannot alter the schema of a table that has enabled, dependent materialized view.

Related Information

Materialized Views
Materialized Views Restrictions
Dropping a Materialized View
ALTER MATERIALIZED VIEW Statement [page 692]
BACKUP DATABASE Statement [page 778]
CREATE MATERIALIZED VIEW Statement [page 896]
sa_refresh_materialized_views System Procedure [page 1681]
sa_materialized_view_info System Procedure [page 1643]
sa_materialized_view_can_be_immediate System Procedure [page 1641]

1.4.4.222 REFRESH TEXT INDEX Statement

Refreshes a text index.

Syntax

```
REFRESH TEXT INDEX text-index-name ON [ owner.]table-name
[ WITH { 
      ISOLATION LEVEL isolation-level
      | EXCLUSIVE MODE
      | SHARE MODE } ]
[ FORCE { BUILD | INCREMENTAL } ]
```

Parameters

WITH clause

Use the WITH clause to specify what kind of locks to obtain on the underlying base tables during the refresh. The types of locks obtained determine how the text index is populated and how concurrency for transactions is affected. If you do not specify the WITH clause, the default is WITH ISOLATION LEVEL READ UNCOMMITTED, regardless of any isolation level set for the connection.

You can specify the following WITH clause options:

ISOLATION LEVEL isolation-level

Use WITH ISOLATION LEVEL to change the isolation level for the execution of the refresh operation.

The original isolation level of the connection is restored at the end of the statement execution.

EXCLUSIVE MODE

Use WITH EXCLUSIVE MODE if you do not want to change the isolation level, but want to guarantee that the data is updated to be consistent with committed data in the underlying table. When using WITH EXCLUSIVE MODE, exclusive table locks are placed on the underlying base table and no other transaction can execute queries, updates, or any other action against the underlying table(s) until the refresh operation is complete. If table locks cannot be obtained, the refresh operation fails and an error is returned.

SHARE MODE

Use WITH SHARE MODE to give read access on the underlying table to other transactions while the refresh operation takes place. When this clause is specified, shared table locks are obtained on the underlying base table before the refresh operation is performed and are held until the refresh operation completes.

FORCE clause

Use this clause to specify the refresh method. If this clause is not specified, the database server decides whether to do an incremental update or a full rebuild based on how much of the table has changed.

FORCE BUILD clause

Refreshes the text index by recreating it. Use this clause to force a complete rebuild of the text index.

FORCE INCREMENTAL clause

Refreshes the text index based only on what has changed in the underlying table. An incremental refresh takes less time to complete if there have not been a significant amount of updates to the underlying table. Use this clause to force an incremental update of the text index.

An incremental refresh does not remove deleted entries from the text index. As a result, the size of the text index may be larger than expected to contain the current and historic data. Typically, this issue occurs with text indexes that are always manually refreshed with the FORCE INCREMENTAL clause. On automatically refreshed text indexes, historic data is automatically deleted when it makes up 50% of the total size of the text index.

Remarks

This statement can only be used on text indexes defined as MANUAL REFRESH or AUTO REFRESH.

When using the FORCE clause, you can examine the results of the sa_text_index_stats system procedure to decide whether a complete rebuild (FORCE BUILD), or incremental update (FORCE INCREMENTAL) is most appropriate.

You cannot execute the REFRESH TEXT INDEX statement on a text index that is defined as IMMEDIATE REFRESH.

For MANUAL REFRESH text indexes, use the sa_text_index_stats system procedure to determine whether the text index should be refreshed. Divide pending_length by doc_length, and use the percentage as a guide for deciding whether a refresh is required. To determine the type of rebuild required, use the same process for deleted_length and doc_count.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

You must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- ALTER ANY INDEX system privilege
- CREATE ANY OBJECT system privilege
- ALTER ANY OBJECT system privilege

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement refreshes a fictitious text index called MarketingTextIndex, forcing it to be rebuilt.

```
REFRESH TEXT INDEX MarketingTextIndex ON GROUPO.MarketingInformation
   FORCE BUILD;
```

Related Information

Text Index Concepts and Reference
Transactions and Isolation Levels
Isolation Levels and Consistency
Table Locks
Snapshot Isolation
Text Index Refresh Types
CREATE TEXT INDEX Statement [page 1030]
ALTER TEXT INDEX Statement [page 762]
DROP TEXT INDEX Statement [page 1136]
TRUNCATE TEXT INDEX Statement [page 1445]
sa_refresh_text_indexes System Procedure [page 1683]
sa_text_index_stats System Procedure [page 1735]

1.4.4.223 REFRESH TRACING LEVEL Statement (Deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The REFRESH TRACING LEVEL statement reloads the tracing levels from the sa_diagnostic_tracing_level table while a tracing session is in progress.

Syntax

REFRESH TRACING LEVEL

Remarks

This statement is used to reload the tracing level information from the sa_diagnostic_tracing_level table. It must be called from a database that is being profiled.

When a tracing session is first started, rows from the sa_diagnostic_tracing_level table are loaded into server memory to control what kind of information is traced. To change the types of data being traced without stopping and restarting the tracing session, manually delete or insert the appropriate rows in the sa_diagnostic_tracing_level table, and then execute the REFRESH TRACING LEVEL statement to reload the settings.

To see the current tracing levels, query the sa_diagnostic_tracing_level table as follows:

```
SELECT * FROM sa_diagnostic_tracing_level WHERE enabled = 1;
```

Suppose you are troubleshooting a performance problem. You turn on a high level of tracing for the entire database to capture the queries that are causing the problem. After starting the tracing session, you find that capturing all queries for all users on your system slows down your database too much, so you decide you would rather limit tracing to one user and wait for that user to report a problem. However, you do not want to stop the tracing session to change the settings.

You can also do this from the command line by replacing the rows in sa_diagnostic_tracing_level table where scope=DATABASE and enabled=1, with equivalent rows where scope=USER, identifier=`userid`, enabled=1, and so on. Then, you execute a REFRESH TRACING LEVEL statement to continue tracing using use the new settings.

Privileges

You must have the MANAGE PROFILING system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example refreshes the tracing level:

```
REFRESH TRACING LEVEL;
```

Related Information

SQL Anywhere Profiler
Diagnostic Tracing (Deprecated)
ATTACH TRACING Statement (Deprecated) [page 775]
DETACH TRACING Statement (Deprecated) [page 1084]
sa_diagnostic_tracing_level Table (Deprecated) [page 1508]

1.4.4.224 RELEASE MUTEX Statement

Releases the specified connection-scope mutex, if it is locked by the current connection.

Syntax

```
RELEASE MUTEX [ owner.]mutex-name
```

Parameters

owner

The owner of the mutex. `owner` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

mutex-name

The name of the mutex. `mutex-name` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

Remarks

The RELEASE MUTEX statement releases one instance of lock on the mutex. So, if a connection has locked the mutex multiple times, then only one lock on the mutex is released per RELEASE MUTEX statement.

An error is returned if the mutex was not locked by the current connection or if the release is being requested for a transaction-scope mutex.

The RELEASE MUTEX statement will succeed on a dropped mutex that is locked by the current connection.

Privileges

You must have the UPDATE ANY MUTEX SEMAPHORE system privilege or be the owner of the mutex.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement releases the protect_my_cr_section mutex:

```
RELEASE MUTEX protect_my_cr_section;
```

Related Information

Mutexes and Semaphores
DROP MUTEX Statement [page 1107]
CREATE MUTEX Statement [page 906]
LOCK MUTEX Statement [page 1265]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.225 RELEASE SAVEPOINT Statement

Releases a savepoint within the current transaction.

Syntax

```
RELEASE SAVEPOINT [ savepoint-name ]
```

Remarks

Release a savepoint. The `savepoint-name` is an identifier specified on a SAVEPOINT statement within the current transaction. If `savepoint-name` is omitted, the most recent savepoint is released.

Releasing a savepoint does not do any type of COMMIT. It simply removes the savepoint from the list of currently active savepoints.

There must have been a corresponding SAVEPOINT within the current transaction.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

RELEASE SAVEPOINT is part of optional ANSI/ISO SQL Language Feature T271, "Savepoints".

Related Information

Savepoints Within Transactions
BEGIN TRANSACTION Statement [T-SQL] [page 792]
COMMIT Statement [page 811]

ROLLBACK Statement [page 1355]
ROLLBACK TO SAVEPOINT Statement [page 1356]
SAVEPOINT Statement [page 1361]

1.4.4.226 REMOTE RESET Statement [SQL Remote]

Starts all subscriptions for a remote user in a single transaction in custom database-extraction procedures.

Syntax

```
REMOTE RESET userid
```

Remarks

This statement starts all subscriptions for a remote user in a single transaction. It sets the log_sent and confirm_sent values in ISYSREMOTEUSER table to the current position in the transaction log. It also sets the created and started values in ISYSSUBSCRIPTION to the current position in the transaction log for all subscriptions for this remote user. The statement does not do a commit. You must do an explicit commit after this call.

To write an extraction process that is safe on a live database, the data must be extracted at isolation level 3 in the same transaction as the subscriptions are started.

This statement is an alternative to start subscription. START SUBSCRIPTION has an implicit commit as a side effect, so that if a remote user has several subscriptions, it is impossible to start them all in one transaction using START SUBSCRIPTION.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

No automatic commit is done by this statement.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement resets the subscriptions for remote user Sam_Singer:

```
REMOTE RESET Sam_Singer;
COMMIT;
```

Related Information

Creating Multiple Remote Databases (Command Line)
Starting Subscriptions (SQL Central)
START SUBSCRIPTION Statement [SQL Remote] [page 1416]
SYSREMOTEUSER System View [page 1937]

1.4.4.227 REMOVE EXTERNAL OBJECT Statement

Removes an external object from the database.

Syntax

```
REMOVE EXTERNAL OBJECT object-name
```

Parameters

object-name

The name of the external object.

Remarks

None.

Privileges

You must be the owner of the external object, or have the MANAGE ANY EXTERNAL OBJECT system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

External Environment Support
ALTER EXTERNAL ENVIRONMENT Statement [page 680]
INSTALL EXTERNAL OBJECT Statement [page 1238]
START EXTERNAL ENVIRONMENT Statement [page 1410]
STOP EXTERNAL ENVIRONMENT Statement [page 1423]
SYSEXTERNENV System View [page 1908]
SYSEXTERNENVOBJECT System View [page 1910]

1.4.4.228 REMOVE JAVA Statement

Removes a class or JAR file from a database.

Syntax

```
REMOVE JAVA
    { CLASS java-class-name [ , java-class-name ... ]
    | JAR jar-name [ , jar-name ... ] }
```

Parameters

CLASS clause

The `java-class-name` parameter is the name of one or more Java class to be removed. These classes must be installed classes in the current database.

JAR clause

The `jar-name` is a single-quoted character string value of maximum length 255. Each `jar-name` must be equal to the `jar-name` of a retained JAR in the current database. Equality of `jar-name` is determined by the character string comparison rules of the SQL system.

Remarks

Removes a class or JAR file from the database. The class or JAR must already be installed.

Privileges

You must be the owner of the class or JAR file, or have the MANAGE ANY EXTERNAL OBJECT system privilege.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example removes a fictitious Java class named Demo from the current database:

```
REMOVE JAVA CLASS Demo;
```

This example removes a fictitious Java JAR named myJar from the current database:

```
REMOVE JAVA JAR 'myJar';
```

Related Information

INSTALL JAVA Statement [page 1240]
SYSJAR System View [page 1918]
SYSJARCOMPONENT System View [page 1919]
SYSJAVACLASS System View [page 1920]

1.4.4.229 REORGANIZE TABLE Statement

Defragments tables when a full rebuild of the database is not possible due to the requirements for continuous access to the database.

Syntax

```
REORGANIZE TABLE [ owner.]table-name
[ { PRIMARY KEY
| FOREIGN KEY foreign-key-name
| INDEX index-name } ]
```

Parameters

Reorganize the table according to the values in one of the following:

PRIMARY KEY clause

Reorganizes the primary key index for the table.

FOREIGN KEY clause

Reorganizes the specified foreign key.

INDEX clause

Reorganizes the specified index.

Remarks

Table fragmentation can impede performance. Use this statement to defragment rows in a table, or to compress indexes which have become sparse due to DELETEs. It may also reduce the total number of pages used to store the table and its indexes, and it may reduce the number of levels in an index tree. However, it will not result in a reduction of the total size of the database file. Use the sa_table_fragmentation and sa_index_density system procedures to select tables worth processing.

If an index or key is not specified, the reorganization process defragments rows in the table by deleting and re-inserting groups of rows. For each group, an exclusive lock on the table is obtained. Once the group has been processed, the lock is released and re-acquired (waiting if necessary), providing an opportunity for other connections to access the table. Checkpoints are suspended while a group is being processed; once a group is finished, a checkpoint may occur. The rows are processed in order of the clustered index if one exists; otherwise, they are processed in order of the primary key. If the table does not have a clustered index or a primary key, an error is returned. The processed rows are re-inserted at the end of the table, resulting in the rows being clustered by primary key at the end of the process. The same amount of work is required, regardless of how fragmented the rows initially were.

If an index or key is specified, the specified index is processed. For the duration of the operation, an exclusive lock is held on the table and checkpoints are suspended. Any attempts to access the table by other connections will block or fail, depending on their setting of the blocking option. The duration of the lock is minimized by pre-reading the index pages before obtaining the exclusive lock.

Since reorganization may modify many pages, the checkpoint log can become large. This can result in an increase in the database file size. However, this increase is temporary since the checkpoint log is deleted at shutdown and the file is truncated at that point.

This statement is not logged to the transaction log.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

Privileges

You must be the owner of the table, or have the REORGANIZE ANY OBJECT system privilege.

Side Effects

Before starting the reorganization, a checkpoint is done to try to maximize the number of free pages. Also, when executing the REORGANIZE TABLE statement, there is an implied commit for approximately every 100 rows, so reorganizing a large table causes multiple commits to take place.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement reorganizes the primary key index for the Employees table:

```
REORGANIZE TABLE GROUPO.Employees
PRIMARY KEY;
```

The following statement reorganizes the table pages of the Employees table:

```
REORGANIZE TABLE GROUPO.Employees;
```

The following statement reorganizes the index IX_product_name on the Products table:

```
REORGANIZE TABLE GROUPO.Products
```

```
    INDEX IX_product_name;
```

The following statement reorganizes the foreign key FK_DepartmentID_DepartmentID for the Employees table:

```
REORGANIZE TABLE GROUPO.Employees
   FOREIGN KEY FK_DepartmentID_DepartmentID;
```

Related Information

Snapshot Isolation
progress_messages Option

1.4.4.230 RESIGNAL Statement [SP]

Resignals an exception condition.

Syntax

```
RESIGNAL [ exception-name ]
```

Remarks

Within an exception handler, RESIGNAL allows you to quit the compound statement with the exception still active, or to quit reporting another named exception. The exception is handled by another exception handler or returned to the application.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The RESIGNAL statement is part of optional ANSI/ISO SQL Language Feature P002, "Computational completeness".

Transact-SQL

The RESIGNAL statement cannot be used in Transact-SQL compound statements and procedures.

Example

The example fragment returns all exceptions except SQLSTATE 52003 to the application.

```
...
DECLARE COLUMN_NOT_FOUND EXCEPTION
   FOR SQLSTATE '52003';
...
EXCEPTION
WHEN COLUMN_NOT_FOUND THEN
SET message='Column not found';
WHEN OTHERS THEN
RESIGNAL;
```

Related Information

Exception Handlers
SIGNAL Statement [SP] [page 1405]
BEGIN Statement [page 784]
RAISERROR Statement [page 1315]

1.4.4.231 RESTORE DATABASE Statement

Restores a backed up database from an archive.

Syntax

```
RESTORE DATABASE filename
FROM archive-root
[ CATALOG ONLY
  | [ RENAME dbspace-name TO new-dbspace-name ] ... ]
[ HISTORY { ON | OFF } ]
[ KEY encryption-key ]
```

```
filename : string | variable
archive-root : string | variable
```

```
new-dbspace-name : string | variable
```

Parameters

FROM clause

Use this clause to specify the location of the backup.

CATALOG ONLY clause

Retrieves information about the named archive, and places it in the backup history file (`backup.syb`), but does not restore any data from the archive.

RENAME clause

Allows you to specify a new location for each dbspace. You cannot use the RENAME clause to change the dbspace name. However, you can use the RENAME clause to change the file name.

HISTORY clause

Allows you to control whether the RESTORE DATABASE operation is recorded in the history file, `backup.syb`.

KEY clause

Allows you to specify the encryption key to restore an archived strongly encrypted database that was backed up with free page elimination on. If the back up was made with free page elimination off, then it is not necessary to specify the encryption key to restore the database. The key can be either a string or a variable name.

As of version 12, you cannot restore archive backups created with version 11 or earlier database servers.

Remarks

Unless HISTORY OFF is specified, each RESTORE DATABASE operation updates a backup history file called `backup.syb`. This file records the BACKUP and RESTORE operations that have been performed on a database server. Consider preventing the RESTORE DATABASE operation from being recorded in `backup.syb` if the following conditions apply:

- your RESTORE DATABASE operations occur frequently
- there is no procedure to periodically archive or delete the `backup.syb` file
- disk space is very limited

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

RESTORE DATABASE replaces the database that is being restored. If you need incremental backups, use the image format of the BACKUP command and save only the transaction log; however.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

You cannot be connected to the database you are restoring. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Predefined Dbspaces
Database Backup and Recovery
BACKUP DATABASE Statement [page 778]
SALOGDIR Environment Variable
-gu Database Server Option
progress_messages Option

1.4.4.232 RESUME Statement

Resumes execution of a cursor that returns result sets.

Syntax

```
RESUME cursor-name
```

```
cursor-name :  identifier | hostvar
```

Remarks

This statement resumes execution of a procedure that returns result sets. The procedure executes until the next result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set. This warning is also set when you RESUME a cursor for a SELECT statement.

The RESUME statement is not supported in Interactive SQL.

The cursor must have been previously opened.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Following are Embedded SQL examples.

```
EXEC SQL RESUME cur_employee;
EXEC SQL RESUME :cursor_var;
```

Related Information

Result Sets
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
FETCH Statement [ESQL] [SP] [page 1162]

1.4.4.233 RETURN Statement

Exits from a function, procedure, or batch unconditionally, optionally providing a return value.

Syntax

```
RETURN [ expression ]
```

Remarks

A RETURN statement causes an immediate exit from a block of SQL. If `expression` is supplied, the value of `expression` is returned as the value of the function or procedure. Subqueries cannot be used in `expression`.

If the RETURN appears inside an inner BEGIN block, it is the outer BEGIN block that is terminated.

Statements following a RETURN statement are not executed.

Within a function, the expression should be of the same data type as the function's RETURNS data type.

Within a procedure, RETURN is used for Transact-SQL compatibility, and is used to return an integer error code.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Core Feature.

Example

The following function returns the product of three numbers:

```
CREATE FUNCTION product (
   a NUMERIC,
   b NUMERIC,
   c NUMERIC )
RETURNS NUMERIC
BEGIN
   RETURN ( a * b * c );
END;
```

Calculate the product of three numbers:

```
SELECT product(2, 3, 4);
```

product(2, 3, 4)
24.000000

The following procedure uses the RETURN statement to avoid executing a complex query if it is meaningless:

```
CREATE PROCEDURE customer_products
( in customer_ID integer DEFAULT NULL)
RESULT ( ID integer, quantity_ordered integer )
BEGIN
   IF customer_ID NOT IN (SELECT ID FROM Customers)
   OR customer_ID IS NULL THEN
      RETURN
   ELSE
      SELECT Products.ID,sum(
         SalesOrderItems.Quantity )
      FROM  GROUPO.Products,
            SalesOrderItems,
            SalesOrders
      WHERE SalesOrders.CustomerID=customer_ID
      AND SalesOrders.ID=SalesOrderItems.ID
      AND SalesOrderItems.ProductID=Products.ID
      GROUP BY Products.ID
   END IF
END;
```

Related Information

Returning a Value Using the RETURN Statement
CREATE FUNCTION Statement [page 861]
CREATE FUNCTION Statement [External Call] [page 866]
CREATE FUNCTION Statement [Web Service] [page 874]
CREATE PROCEDURE Statement [page 913]
CREATE PROCEDURE Statement [External Call] [page 921]
CREATE PROCEDURE Statement [Web Service] [page 931]
BEGIN Statement [page 784]

1.4.4.234 REVOKE Statement

Revokes system and object-level privileges from users and roles.

Syntax

Revoke system privileges

```
REVOKE [ { EXERCISE | ADMIN } OPTION FOR ] privilege
FROM grantee, ...
```

```
grantee :
{ system-role | userid }
```

Revoke object-level privileges

```
REVOKE object-level-privilege[,...]
ON [ owner.]table-or-view
FROM userid[,...]
```

```
object-level-privilege :
ALL [ PRIVILEGES ]
| ALTER
| DELETE
| INSERT
| LOAD
| REFERENCES [ ( column-name[,...] ) ]
| SELECT [ ( column-name[,...] ) ]
| TRUNCATE
| UPDATE [ ( column-name[,...] ) ]
```

Revoke CONNECT, INTEGRATED LOGIN, and KERBEROS LOGIN

```
REVOKE capability FROM userid[,...]
```

```
capability :
CONNECT
| INTEGRATED LOGIN
| KERBEROS LOGIN
```

Revoke EXECUTE on a procedure

```
REVOKE EXECUTE
ON [ owner.]procedure-name[,...]
FROM userid[,...]
```

Revoke USAGE on a sequence

```
REVOKE USAGE ON SEQUENCE sequence-name[,...]
FROM userid[,...]
```

Parameters

{ EXERCISE | ADMIN } OPTION FOR clause

Specify the ADMIN OPTION FOR clause to revoke administration rights for the privilege, but leave exercise rights. Specify the EXERCISE OPTION FOR clause to revoke exercise rights for the privilege, but leave administration rights. If the clause is not specified, both rights are revoked.

REVOKE CONNECT

REVOKE CONNECT removes a user ID from a database, and also destroys any objects (tables, views, procedures, and so on) owned by that user. However, it is recommended that you use the DROP USER statement to remove users instead of the REVOKE CONNECT statement. System privileges granted by the user remain in effect; however, object-level privileges granted by the user are revoked.

You cannot execute a REVOKE CONNECT statement on a user if the user being dropped owns a table referenced by a view owned by another user.

When you are connected to the utility database, executing REVOKE CONNECT FROM DBA disables future connections to the utility database. No future connections can be made to the utility database unless you use a connection that existed before the REVOKE CONNECT was executed, or restart the database server.

REVOKE USAGE ON SEQUENCE

Specify this syntax to remove the privilege to evaluate the current or next value in a sequence.

Remarks

If a privilege that is being revoked was not granted to `grantee`, then the statement does nothing, and does not return an error.

REVOKE fails with an error if, as a consequence of executing the statement, the number of administrators for the system privilege being revoked would fall below the required minimum as set by the min_role_admins database option.

When you revoke an object-level privilege for a user who also had administrative rights for that privilege, then everyone who that user granted the privilege to also has their privilege revoked, as well as anyone that the grantees granted it to, and so on.

If you are revoking connection-related privileges from a user, the user must not be connected to the database.

When revoking a system privilege from the UPGRADE ROLE system privilege after an upgrade, you must use the special internal representation SYS_UPGRADE_ROLE_ROLE. For example, `REVOKE privilege-name FROM SYS_UPGRADE_ROLE_ROLE;`).

The REVOKE syntax related to authorities, permissions, and groups used in pre-16.0 versions of the software is still supported but deprecated.

Privileges

You must have administration rights for the system privileges that you are revoking.

If you are revoking object-level privileges, you must have one of the following:

- Ownership of the object
- Administrative rights on the object-level privilege for that object

- MANAGE ANY OBJECT PRIVILEGE system privilege

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

REVOKE `capability` is not part of the standard. REVOKE `object-level-privilege` and REVOKE EXECUTE are Core Features of the ANSI/ISO SQL Standard. With REVOKE ALL (revoking all object-level privileges), the PRIVILEGES keyword is optional, while in the Standard it is mandatory.

REVOKE USAGE ON SEQUENCE is part of optional ANSI/ISO SQL Language Feature T176, "Sequence generator support".

Example

This example prevents user Dave from updating the Employees table.

```
REVOKE UPDATE ON GROUPO.Employees FROM Dave;
```

This example prevents a fictitious user-extended role called Finance from executing the procedure ShowCustomers.

```
REVOKE EXECUTE ON ShowCustomers FROM Finance;
```

This example drops user FranW from the database. This syntax is deprecated; consider using the DROP USER statement instead.

```
REVOKE CONNECT FROM FranW;
```

This example revokes database login privilege from a fictitious Kerberos user, pchin.

```
REVOKE KERBEROS LOGIN
FROM "pchin@MYREALM.COM";
```

Related Information

System Privileges
Object-level Privileges
Viewing the Roles and Privileges for a User or Role (SQL Central)

Distributing Privileges Granted to the UPGRADE ROLE System Privilege After an Upgrade
min_role_admins Option
REVOKE ROLE Statement [page 1352]

1.4.4.235 REVOKE CONSOLIDATE Statement [SQL Remote]

Stops a consolidated database from receiving SQL Remote messages from this database.

Syntax

```
REVOKE CONSOLIDATE FROM userid
```

Remarks

CONSOLIDATE privileges must be granted at a remote database for the user ID representing the consolidated database. The REVOKE CONSOLIDATE statement removes the consolidated database user ID from the list of users receiving messages from the current database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit. Drops all subscriptions for the user.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

- The following statement revokes consolidated status from the Sam_Singer user ID:

```
REVOKE CONSOLIDATE FROM Sam_Singer;
```

Related Information

GRANT CONSOLIDATE Statement [SQL Remote] [page 1199]
GRANT ROLE Statement [page 1210]
REVOKE PUBLISH Statement [SQL Remote] [page 1349]
REVOKE REMOTE Statement [SQL Remote] [page 1351]

1.4.4.236 REVOKE PUBLISH Statement [SQL Remote]

Terminates the identification of the named user ID as the current publisher. You must have the SYS_REPLICATION_ADMIN_ROLE system role to revoke publisher rights.

Syntax

```
REVOKE PUBLISH FROM userid
```

Remarks

Each database in a SQL Remote installation is identified in outgoing messages by a publisher user ID. The current publisher user ID can be determined by querying the CURRENT PUBLISHER special value as follows:

```
SELECT CURRENT PUBLISHER;
```

The REVOKE PUBLISH statement ends the identification of the named user ID as the publisher. To change publishers, you must REVOKE PUBLISH from the current publisher, and then GRANT PUBLISH to the new publisher.

If you change the publisher user ID at any consolidated or remote database in a SQL Remote installation, you must ensure that the new publisher user ID is granted REMOTE privilege on all databases receiving messages from the database. Making this change requires all subscriptions to be dropped and recreated.

Do not revoke the publisher while the database has active SQL Remote publications or subscriptions.

Revoking publisher and not granting it to a new user has consequences for a SQL Remote installation:

- You cannot insert data into any tables with a CURRENT PUBLISHER column as part of the primary key. Any outgoing messages are not identified with a publisher user ID, and so are not accepted by recipient databases.

Executing this statement changes the value of the PUBLIC.db_publisher database option.

Privileges

You must have the SET ANY SYSTEM OPTION system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Terminate the identification of publisher_ID as the current publisher.

```
REVOKE PUBLISH FROM publisher_ID;
```

Related Information

GRANT ROLE Statement [page 1210]
db_publisher Option
GRANT PUBLISH Statement [SQL Remote] [page 1206]
REVOKE REMOTE Statement [SQL Remote] [page 1351]
REVOKE CONSOLIDATE Statement [SQL Remote] [page 1348]
CURRENT PUBLISHER Special Value [page 91]

1.4.4.237 REVOKE REMOTE Statement [SQL Remote]

Stops a user from being able to receive SQL Remote messages from this database.

Syntax

```
REVOKE REMOTE FROM userid, ...
```

Remarks

REMOTE privilege is required for a user ID to receive messages in a SQL Remote replication installation. The REVOKE REMOTE statement removes a user ID from the list of users receiving messages from the current database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit. Drops all subscriptions for the user.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

```
REVOKE REMOTE FROM Sam_Singer;
```

Related Information

Revoking REMOTE Privilege (SQL Central)
REVOKE ROLE Statement [page 1352]
REVOKE PUBLISH Statement [SQL Remote] [page 1349]
GRANT REMOTE Statement [SQL Remote] [page 1208]
REVOKE CONSOLIDATE Statement [SQL Remote] [page 1348]

1.4.4.238 REVOKE ROLE Statement

Revokes roles and privileges from users and roles.

Syntax

Revoke system roles

```
REVOKE ROLE system-role
FROM grantee, ...
```

```
grantee :
{ system-role | userid }
```

```
system-role :
dbo
| DIAGNOSTICS
| PUBLIC
| rs_systabgroup
| SA_DEBUG
| SYS
| SYS_REPLICATION_ADMIN_ROLE
| SYS_RUN_REPLICATION_ROLE
| SYS_SAMONITOR_ADMIN_ROLE
| SYS_SPATIAL_ADMIN_ROLE
```

Revoke user-defined roles

```
REVOKE [ { EXERCISE | ADMIN } OPTION FOR ] ROLE user-defined-role
FROM grantee, ...
```

```
grantee :
{ system-role | userid }
```

Revoke compatibility roles

```
REVOKE [ { EXERCISE | ADMIN } OPTION FOR ] ROLE compatibility-role-name
FROM grantee, ...
```

```
compatibility-role-name :
SYS_AUTH_BACKUP_ROLE
| SYS_AUTH_DBA_ROLE
| SYS_AUTH_PROFILE_ROLE
| SYS_AUTH_READCLIENTFILE_ROLE
| SYS_AUTH_READFILE_ROLE
```

```
    | SYS_AUTH_RESOURCE_ROLE
    | SYS_AUTH_SA_ROLE
    | SYS_AUTH_SSO_ROLE
    | SYS_AUTH_VALIDATE_ROLE
    | SYS_AUTH_WRITECLIENTFILE_ROLE
    | SYS_AUTH_WRITEFILE_ROLE
```

```
grantee :
{ system-role | userid }
```

Parameters

{ EXERCISE | ADMIN } OPTION FOR clause

Specify the ADMIN OPTION FOR clause to revoke administration rights for the role, but leave exercise rights. Specify the EXERCISE OPTION FOR clause to revoke exercise rights for the role, but leave administration rights. If the clause is not specified, both rights are revoked.

Remarks

If a role that is being revoked was not granted to `grantee`, then the statement does nothing, and does not return an error.

REVOKE ROLE fails with an error if, as a consequence of executing the statement, the number of administrators for the role being revoked would fall below the required minimum as set by the min_role_admins database option.

When revoking a role from the MANAGE ROLES system privilege, you must use the special internal representation SYS_MANAGE_ROLES_ROLE. For example, `REVOKE ROLE role-name FROM SYS_MANAGE_ROLES_ROLE;`.

The REVOKE syntax related to authorities, permissions, and groups used in pre-16.0 versions of the software is still supported but deprecated.

Privileges

You must have administration rights for the role that you are revoking.

To revoke the SYS_RUN_REPLICATION_ROLE system role, you must have the SYS_REPLICATION_ADMIN_ROLE.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example revokes the SYS_AUTH_VALIDATE_ROLE compatibility role from fictitious user Jim.

```
REVOKE ROLE SYS_AUTH_VALIDATE_ROLE FROM Jim;
```

This example revokes the DIAGNOSTICS system role from a fictitious user named Administrator.

```
REVOKE ROLE DIAGNOSTICS FROM Administrator;
```

The following statement revokes SYS_REPLICATION_ADMIN_ROLE from user Sam_Singer.

```
REVOKE ROLE SYS_REPLICATION_ADMIN_ROLE FROM Sam_Singer;
```

The following statement revokes SYS_RUN_REPLICATION_ROLE from user Sam_Singer.

```
REVOKE ROLE SYS_RUN_REPLICATION_ROLE FROM Sam_Singer;
```

Related Information

Role Administrators
Replication-related System Roles
Roles
Initiation of Synchronization
Viewing the Roles and Privileges for a User or Role (SQL Central)
min_role_admins Option
REVOKE Statement [page 1345]
REVOKE PUBLISH Statement [SQL Remote] [page 1349]
REVOKE REMOTE Statement [SQL Remote] [page 1351]
REVOKE CONSOLIDATE Statement [SQL Remote] [page 1348]
REVOKE Statement (Authorities and Groups) (Deprecated)

1.4.4.239 ROLLBACK Statement

Ends a transaction and undo any changes made since the last COMMIT or ROLLBACK.

Syntax

```
ROLLBACK [ WORK ]
```

Remarks

A transaction is the logical unit of work done on one database connection to a database between COMMIT or ROLLBACK statements. The ROLLBACK statement ends the current transaction and undoes all changes made to the database since the previous COMMIT or ROLLBACK.

Privileges

None.

Side Effects

Closes all cursors not opened WITH HOLD.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Interactive SQL Options
Cancellation of Changes
COMMIT Statement [page 811]
Keyboard Shortcuts (Interactive SQL)
ROLLBACK TO SAVEPOINT Statement [page 1356]

1.4.4.240 ROLLBACK TO SAVEPOINT Statement

Cancels any changes made since a SAVEPOINT.

Syntax

```
ROLLBACK TO SAVEPOINT [ savepoint-name ]
```

Remarks

The ROLLBACK TO SAVEPOINT statement will undo any changes that have been made since the SAVEPOINT was established. Changes made before the SAVEPOINT are not undone; they are still pending.

The `savepoint-name` is an identifier that was specified on a SAVEPOINT statement within the current transaction. If `savepoint-name` is omitted, the most recent savepoint is used. Any savepoints since the named savepoint are automatically released.

There must have been a corresponding SAVEPOINT within the current transaction.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

ROLLBACK TO SAVEPOINT is part of optional ANSI/ISO SQL Language Feature T271.

Related Information

Savepoints Within Transactions
BEGIN TRANSACTION Statement [T-SQL] [page 792]

COMMIT Statement [page 811]
RELEASE SAVEPOINT Statement [page 1331]
ROLLBACK Statement [page 1355]
SAVEPOINT Statement [page 1361]

1.4.4.241 ROLLBACK TRANSACTION Statement [T-SQL]

Cancels any changes made since a SAVE TRANSACTION.

Syntax

```
ROLLBACK TRANSACTION [ savepoint-name ]
```

Remarks

The ROLLBACK TRANSACTION statement undoes any changes that have been made since a savepoint was established using SAVE TRANSACTION. Changes made before the SAVE TRANSACTION are not undone; they are still pending.

The `savepoint-name` is an identifier that was specified on a SAVE TRANSACTION statement within the current transaction. If `savepoint-name` is omitted, all outstanding changes are rolled back. Any savepoints since the named savepoint are automatically released.

There must be a corresponding SAVE TRANSACTION within the current transaction.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example displays five rows with values 10, 20, and so on. The effect of the DELETE, but not the prior INSERTs or UPDATE, is undone by the ROLLBACK TRANSACTION statement.

```
BEGIN
    SELECT row_num INTO #tmp
    FROM sa_rowgenerator( 1, 5 )
    UPDATE #tmp SET row_num=row_num*10
    SAVE TRANSACTION before_delete
    DELETE FROM #tmp WHERE row_num >= 3
    ROLLBACK TRANSACTION before_delete
    SELECT * FROM #tmp
END
```

Related Information

ROLLBACK TO SAVEPOINT Statement [page 1356]
BEGIN TRANSACTION Statement [T-SQL] [page 792]
COMMIT Statement [page 811]
SAVE TRANSACTION Statement [T-SQL] [page 1359]

1.4.4.242 ROLLBACK TRIGGER Statement

Undoes any changes made in a trigger.

Syntax

```
ROLLBACK TRIGGER [ WITH raiserror-statement ]
```

Remarks

The ROLLBACK TRIGGER statement rolls back the work done in a trigger, including the data manipulation that caused the trigger to fire.

Optionally, a RAISERROR statement can executed. If a RAISERROR statement is executed, an error is returned to the application. If no RAISERROR statement is executed, no error is returned.

If a ROLLBACK TRIGGER statement is used within a nested trigger and without a RAISERROR statement, only the innermost trigger and the statement which caused it to fire are undone.

Privileges

None.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Transact-SQL

ROLLBACK TRIGGER is supported in both Watcom SQL and Transact-SQL stored procedures. ROLLBACK TRIGGER is supported by Adaptive Server Enterprise.

Related Information

CREATE TRIGGER Statement [page 1036]
ROLLBACK Statement [page 1355]
ROLLBACK TO SAVEPOINT Statement [page 1356]
RAISERROR Statement [page 1315]

1.4.4.243 SAVE TRANSACTION Statement [T-SQL]

Establishes a savepoint within the current transaction.

Syntax

```
SAVE TRANSACTION savepoint-name
```

Remarks

Establish a savepoint within the current transaction. The `savepoint-name` is an identifier that can be used in a ROLLBACK TRANSACTION statement. All savepoints are automatically released when a transaction ends.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example displays five rows with values 10, 20, and so on. The effect of the DELETE, but not the prior INSERTs or UPDATE, is undone by the ROLLBACK TRANSACTION statement.

```
BEGIN
    SELECT row_num INTO #tmp
    FROM sa_rowgenerator( 1, 5 )
    UPDATE #tmp SET row_num=row_num*10
    SAVE TRANSACTION before_delete
    DELETE FROM #tmp WHERE row_num >= 3
    ROLLBACK TRANSACTION before_delete
    SELECT * FROM #tmp
END
```

Related Information

Savepoints Within Transactions
SAVEPOINT Statement [page 1361]
BEGIN TRANSACTION Statement [T-SQL] [page 792]
COMMIT Statement [page 811]
ROLLBACK TRANSACTION Statement [T-SQL] [page 1357]

1.4.4.244 SAVEPOINT Statement

Establishes a savepoint within the current transaction.

Syntax

```
SAVEPOINT [ savepoint-name ]
```

Remarks

Establish a savepoint within the current transaction. The `savepoint-name` is an identifier that can be used in a RELEASE SAVEPOINT or ROLLBACK TO SAVEPOINT statement. All savepoints are automatically released when a transaction ends.

Savepoints that are established while a trigger or atomic compound statement is executing are automatically released when the atomic operation ends.

You cannot modify data in a proxy table from within a savepoint.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The SAVEPOINT statement is part of optional ANSI/ISO SQL Language Feature T271, "Savepoints".

Transact-SQL

In Transact-SQL, creating a savepoint is accomplished with the SAVE TRANSACTION statement.

Related Information

Savepoints Within Transactions

RELEASE SAVEPOINT Statement [page 1331]
ROLLBACK TO SAVEPOINT Statement [page 1356]
SAVE TRANSACTION Statement [T-SQL] [page 1359]

1.4.4.245 SELECT Statement

Retrieves information from the database.

Syntax

```
[ WITH temporary-views ]
  SELECT [ ALL | DISTINCT ] [ row-limitation-option-1 ] select-list
[ INTO { hostvar-list | variable-list | table-name } ]
[ INTO LOCAL TEMPORARY TABLE { table-name } ]
[ INTO TABLE table-name ]
[ INTO VARIABLE variable-list ]
[ FROM from-expression ]
[ WHERE search-condition ]
[ GROUP BY group-by-expression ]
[ HAVING search-condition ]
[ WINDOW window-expression ]
[ ORDER BY { expression | integer } [ ASC | DESC ], ... ]
[ FOR READ ONLY | for-update-clause ]
[ FOR XML xml-mode ]
[ FOR JSON json-mode ]
[ row-limitation-option-2 ]
[ OPTION( query-hint, ... ) ]
```

```
temporary-views :
  regular-view, ...
| RECURSIVE { regular-view | recursive-view }, ...
```

```
regular-view :
  view-name [ ( column-name, ... ) ]
  AS ( query-block )
```

```
recursive-view :
  view-name ( column-name, ... )
  AS ( initial-query-block UNION ALL recursive-query-block )
```

```
query-block : see the documentation on common elements in SQL syntax
```

```
row-limitation-option-1 :
FIRST
| TOP { ALL | limit-expression } [ START AT startat-expression ]
```

```
row-limitation-option-2 :
LIMIT { [ offset-expression, ] limit-expression | limit-expression OFFSET
offset-expression }
```

```
limit-expression : simple-expression
```

```
startat-expression : simple-expression
```

```
offset-expression : simple-expression
```

```
simple-expression :
integer
| variable
| ( simple-expression )
| ( simple-expression { + | - | * } simple-expression )
```

```
select-list :
expression [ [ AS ] alias-name ], ...
| *
| window-function OVER { window-name | window-spec }
   [ [ AS ] alias-name ]
|  sequence-expression
```

```
sequence-expression :
sequence-name . [ CURRVAL | NEXTVAL ]
```

```
window-function :
RANK( )
| DENSE_RANK( )
| PERCENT_RANK( )
| CUME_DIST( )
| ROW_NUMBER( )
| aggregate-function
```

```
for-update-clause
FOR UPDATE
| FOR UPDATE cursor-concurrency
| FOR UPDATE OF [ ( column-name, ... ) ]
```

```
 cursor-concurrency :
BY { VALUES | TIMESTAMP | LOCK }
```

```
xml-mode :
RAW [ , ELEMENTS ]
| AUTO [ , ELEMENTS ]
| EXPLICIT
```

```
json-mode :
AUTO | EXPLICIT | RAW
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| FORCE NO OPTIMIZATION
| option-name=option-value
```

```
option-name : identifier
```

```
option-value :
hostvar (indicator allowed)
| string
| identifier
| number
```

```
from-expression : see the FROM clause topics
group-by-expression : see the GROUP BY clause topic
window-expression : see the WINDOW clause topic
```

Parameters

WITH or WITH RECURSIVE clause

Define one or more common table expressions, also known as temporary views, to be used elsewhere in the remainder of the statement. These expressions may be non-recursive, or may be self-recursive. Recursive common table expressions may appear alone, or intermixed with non-recursive table expressions, only if the RECURSIVE keyword is specified. Mutually recursive common table expressions are not supported.

This clause is permitted only if the SELECT query block appears in one of the following locations:

- Within a top-level SELECT query block including the top-level SELECT query block of a view definition
- Within a top-level SELECT statement within an INSERT query block
- Within a nested SELECT query block defining a derived table in any type of SQL statement

Recursive expressions consist of an initial subquery and a recursive subquery. The initial-query implicitly defines the schema of the view. The recursive subquery must contain a reference to the view within the FROM clause. During each iteration, this reference refers only to the rows added to the view in the previous iteration. The reference must not appear on the null-supplying side of an outer join. A recursive common table expression must not use aggregate functions and must not contain a GROUP BY, ORDER BY, or DISTINCT clause.

The WITH clause is not supported with remote tables. The WITH clause may also be used in INTERSECT, UNION, and EXCEPT query blocks.

This functionality is available only in the Watcom SQL dialect.

```
WITH RECURSIVE
  manager ( EmployeeID, ManagerID,
            GivenName, Surname, mgmt_level ) AS
( ( SELECT EmployeeID, ManagerID,       -- initial subquery
           GivenName, Surname, 0
    FROM Employees AS e
    WHERE ManagerID = EmployeeID )
  UNION ALL
  ( SELECT e.EmployeeID, e.ManagerID,   -- recursive subquery
           e.GivenName, e.Surname, m.mgmt_level + 1
    FROM Employees AS e JOIN manager AS m
     ON   e.ManagerID =  m.EmployeeID
      AND e.ManagerID <> e.EmployeeID
      AND m.mgmt_level < 20 ) )
SELECT 'Manager', * FROM manager
WHERE mgmt_level > 0
UNION ALL
SELECT 'Employee', * FROM manager
WHERE mgmt_level = 0
ORDER BY mgmt_level, Surname, GivenName;
```

ALL or DISTINCT clause

ALL (the default) returns all rows that satisfy the clauses of the SELECT statement. If DISTINCT is specified, duplicate output rows are eliminated. Many statements take significantly longer to execute when DISTINCT is specified, so reserve use of DISTINCT for cases where it is necessary.

row-limitation clauses

The row limitation clauses allow you to return only a subset of the rows that satisfy the WHERE clause. Only one `row-limitation` clause can be specified at a time. When specifying these clauses, an ORDER BY clause is required to order the rows in a meaningful manner.

row-limitation-option-1

The TOP and START AT arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The TOP argument must evaluate to a value greater than or equal to 0. The START AT argument must evaluate to a value greater than 0.

If `startat-expression` is not specified, the default is 1. If the argument of TOP is ALL, all rows starting at `startat-expression` are returned. The `TOP limit-expression START AT startat-expression` clause is equivalent to `LIMIT ( startat-expression -1 ), limit-expression` or `LIMIT limit-expression OFFSET ( startat-expression -1 )`.

row-limitation-option-2

The LIMIT and OFFSET arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The LIMIT argument must evaluate to a value greater than or equal to 0. The OFFSET argument must evaluate to a value greater than or equal to 0. If `offset-expression` is not specified, the default is 0.

The row limitation clause `LIMIT offset-expression, limit-expression` is equivalent to `LIMIT limit-expression OFFSET offset-expression`. Both of these constructs are equivalent to `TOP limit-expression START AT ( offset-expression + 1 )`.

The LIMIT keyword is disabled by default. Use the reserved_keywords option to enable the LIMIT keyword.

select-list clause

The `select-list` is a list of expressions, separated by commas, specifying what is retrieved from the database. An asterisk (*) means select all columns of all tables in the FROM clause.

Aggregate functions are allowed in the `select-list`. Subqueries are also allowed in the `select-list`. Each subquery must be within parentheses.

Alias names can be used throughout the query to represent the aliased expression.

Alias names are also displayed by Interactive SQL at the top of each column of output from the SELECT statement. If the optional alias name is not specified after an expression, Interactive SQL displays the expression.

> **i** Note
>
> The following characters are not permitted in aliases:
>
> - Double quotes
> - Control characters (any character less than 0X20)
> - Backslashes
> - Square brackets

- Back quotes

INTO clause

Following are the three uses of the INTO clause:

INTO hostvar-list clause

This clause is used in Embedded SQL only. It specifies where the results of the SELECT statement go. There must be one host variable item for each item in the `select-list`. `select-list` items are put into the host variables in order. An indicator host variable is also allowed with each host variable, so the program can tell if the `select-list` item was NULL.

If the query results in no rows being selected, the variables are not updated, and a row not found warning appears.

INTO variable-list clause

This clause is used in procedures and triggers only. It specifies where the results of the SELECT statement go. There must be one variable for each item in the `select-list`. `select-list` items are put into the variables in order.

INTO table-name clause

This clause is used to create a table and fill it with data.

For tables to be created, the query must satisfy one of the following conditions:

- The `table-name` is a temporary table name or is specified as **`owner.`**`table-name`.
- The `select-list` contains a * ore more than one item.

To create a permanent table with one column, the table name must be specified as `owner.table`.

This statement causes a COMMIT before execution as a side effect of creating the table. No privileges are granted on the new table: the statement is a short form for CREATE TABLE followed by INSERT...SELECT.

Tables created using this clause do not have a primary key defined. You can add a primary key using ALTER TABLE. A primary key should be added before applying any updates or deletes to the table; otherwise, these operations result in all column values being logged in the transaction log for the affected rows.

INTO LOCAL TEMPORARY TABLE clause

This clause is used to create a local, temporary table and populate it with the results of the query. When you use this clause, it is not necessary to start the temporary table name with #.

INTO TABLE table-name clause

This clause always creates a table and populates it with the results of the query. The INTO TABLE clause behaves the same way as the INTO `table-name` clause, with the exception that the query does not need to satisfy the `select-list` conditions of the INTO `table-name` clause.

INTO VARIABLE variable-list clause

This clause specifies where the results of the SELECT statement go. The values returned by the query are assigned to the fields of the row variable in order.

If there is a single item in the INTO VARIABLE clause and multiple items in the `select-list`, then the single item is interpreted as a row variable and the returned column values are assigned to the fields of the

row variable in order. If the INTO VARIABLE clause contains more than one item, then the INTO `variable-list` semantics are applied.

> **i** Note
>
> The row variable cannot include another composite data type, such as a row or array, and it must be composed as a collection of simple variables; otherwise, an error is returned.

FROM clause

Rows are retrieved from the tables and views specified in the `table-expression`. A SELECT statement with no FROM clause can be used to display the values of expressions not derived from tables. For example, these two statements are equivalent and display the value of the global variable @@version.

```
SELECT @@version;
SELECT @@version FROM SYS.DUMMY;
```

WHERE clause

This clause specifies which rows are selected from the tables named in the FROM clause. It can be used to do joins between multiple tables, as an alternative to the ON phrase (which is part of the FROM clause).

GROUP BY clause

You can group by columns, alias names, or functions. The result of the query contains one row for each distinct set of values in the named columns, aliases, or functions. As with DISTINCT and the set operations UNION, INTERSECT, and EXCEPT, the GROUP BY clause treats NULL values in the same manner as any other value in each domain. In other words, multiple NULL values in a grouping attribute form a single group. Aggregate functions can then be applied to these groups to get meaningful results.

When GROUP BY is used, the `select-list`, HAVING clause, and ORDER BY clause must not reference any identifier that is not named in the GROUP BY clause. The exception is that the `select-list` and HAVING clause can contain aggregate functions.

HAVING clause

This clause selects rows based on the group values and not on the individual row values. The HAVING clause can only be used if either the statement has a GROUP BY clause or the `select-list` consists solely of aggregate functions. Any column names referenced in the HAVING clause must either be in the GROUP BY clause or be used as a parameter to an aggregate function in the HAVING clause.

WINDOW clause

This clause defines all or part of a window for use with window functions such as AVG and RANK.

ORDER BY clause

This clause sorts the results of a query. Each item in the ORDER BY list can be labeled as ASC for ascending order (the default) or DESC for descending order. If the expression is an integer `n`, then the query results are sorted by the `n`th item in the `select-list`.

The only way to ensure that rows are returned in a particular order is to use ORDER BY. In the absence of an ORDER BY clause, the database server returns rows in whatever order is most efficient. The appearance of result sets may vary depending on when you last accessed the row and other factors.

Items in the ORDER BY clause cannot be table reference variables (variables defined as type TABLE REF).

In Embedded SQL, the SELECT statement is used for retrieving results from the database and placing the values into host variables via the INTO clause. The SELECT statement must return only one row. For multiple row queries, you must use cursors.

FOR UPDATE or FOR READ ONLY clause

These clauses specify whether updates are allowed through a cursor opened on the query, and if so, what concurrency semantics can be used. This clause cannot be used with the FOR XML clause.

If you do not use a FOR clause in the SELECT statement, the updatability of a cursor depends on the cursor's declaration and how cursor concurrency is specified by the API. In ODBC, JDBC, OLE DB, ADO.NET, and Embedded SQL, statement updatability is explicit and a read-only cursor is used unless it is overridden by the application. In Open Client and within stored procedures, cursor updatability does not have to be specified, and the default is FOR UPDATE.

For Open Client and stored procedures, cursor updatability and statement updatability is dependent on the setting of the ansi_update_constraints database option and the specific characteristics of the statement, including whether the statement contains ORDER BY, DISTINCT, GROUP BY, HAVING, UNION, aggregate functions, joins, or non-updatable views. For stored procedures, cursors default to FOR UPDATE for single-table queries without an ORDER BY clause, or if the ansi_update_constraints option is set to Off. When the ansi_update_constraints option is set to Cursors or Strict, then cursors over a query containing an ORDER BY clause default to READ ONLY. However, you can explicitly mark cursors as updatable using the FOR UPDATE clause. Because it is expensive to allow updates over cursors with an ORDER BY clause or a join, cursors over a query containing a join of two or more tables are READ ONLY and cannot be made updatable unless the ansi_update_constraints database option is Off.

A cursor declared FOR READ ONLY cannot be used in UPDATE (positioned), DELETE (positioned), or PUT statements. FOR READ ONLY is the default for Embedded SQL.

The FOR UPDATE clause explicitly makes a cursor updatable. The use of FOR UPDATE alone does not, by itself, affect concurrency control on the rows in the result set of the statement. To do this, you must specify either FOR UPDATE BY LOCK or FOR UPDATE BY [VALUES | TIMESTAMP].

FOR UPDATE BY LOCK clause

The database server acquires intent row locks on fetched rows of the result set. These are long-term locks that are held until the transaction is committed or rolled back. Intent row locks are not acquired if the SELECT statement uses an INTO clause since no positioned update can be performed.

FOR UPDATE BY { VALUES | TIMESTAMP } clause

When you specify FOR UPDATE BY TIMESTAMP or FOR UPDATE BY VALUES, the database server uses optimistic concurrency by using a keyset-driven (value-sensitive) cursor. In this situation, lost updates can occur if the application modifies a row outside of the cursor (using a separate statement) or if the application does not heed the warnings and/or errors generated by the server indicating that the row has been modified by another connection.

To ensure that a statement acquires an intent lock, you must do one of the following:

- specify FOR UPDATE BY LOCK in the query
- specify HOLDLOCK, WITH (HOLDLOCK), WITH (UPDLOCK), or WITH (XLOCK) in the FROM clause of the query
- open the cursor with API calls that specify CONCUR_LOCK
- fetch the rows with attributes indicating fetch for update

The FOR UPDATE OF clause explicitly names the columns that can be modified with an UPDATE (positioned), DELETE (positioned), or PUT statement. You cannot use this clause in combination with any other FOR UPDATE, FOR READ ONLY, or FOR XML clause.

FOR UPDATE OF column-list clause

When you specify the FOR UPDATE OF clause, the database server restricts the columns that can be modified with a positioned UPDATE or positioned DELETE statement to those columns that are explicitly named in that clause. An attempt to modify another column results in an error indicating that the column cannot be found. No check is made to determine if a column referenced within the list actually exists, or if that column's table is an updatable table.

FOR XML clause

This clause specifies that the result set is to be returned as an XML document. The format of the XML depends on the mode you specify. This clause cannot be used with the FOR UPDATE or FOR READ ONLY clause. Cursors declared with FOR XML are implicitly READ ONLY.

When you specify RAW mode, each row in the result set is represented as an XML <row> element, and each column is represented as an attribute of the <row> element.

AUTO mode returns the query results as nested XML elements. Each table referenced in the `select-list` is represented as an element in the XML. The order of nesting for the elements is based on the order that tables are referenced in the `select-list`.

EXPLICIT mode allows you to control the form of the generated XML document. Using EXPLICIT mode offers more flexibility in naming elements and specifying the nesting structure than either RAW or AUTO mode.

FOR JSON clause

This clause specifies that the result set is to be returned in JSON format. The JSON format depends on the mode you specify. This clause cannot be used with the FOR UPDATE or FOR READ ONLY clause. Cursors declared with FOR JSON are implicitly READ ONLY.

When you specify RAW mode, each row in the result set is returned as a flattened JSON representation.

AUTO mode returns the query results as nested JSON objects based on query joins.

EXPLICIT mode allows you to control the form of the generated JSON objects. Using EXPLICIT mode offers more flexibility in specifying columns and nested hierarchical objects to produce uniform or heterogeneous arrays.

OPTION clause

This clause provides hints about how to process the query. The following query hints are supported:

MATERIALIZED VIEW OPTIMIZATION clause

Use the MATERIALIZED VIEW OPTIMIZATION clause to specify how the optimizer should make use of materialized views when processing the query. The specified `option-value` overrides the materialized_view_optimization database option for this query only. Possible values for `option-value` are the same values available for the materialized_view_optimization database option.

FORCE OPTIMIZATION clause

When a query specification contains only simple queries (single-block, single-table queries that contain equality conditions in the WHERE clause that uniquely identify a specific row), it typically bypasses cost-based optimization during processing. Sometimes you may want cost-based optimization to occur. For example, if you want materialized views to be considered during query processing, view matching must occur. However, view matching only occurs during cost-based optimization. If you want cost-based optimization to occur for a query, but your query specification contains only simple queries, specify the FORCE OPTIMIZATION option to ensure that the optimizer performs cost-based optimization on the query.

Similarly, specifying the FORCE OPTIMIZATION option in a SELECT statement inside of a procedure forces the use of the optimizer for any call to the procedure. In this case, plans for the statement are not cached.

FORCE NO OPTIMIZATION clause

Specify the FORCE NO OPTIMIZATION clause if you want the statement to bypass the optimizer. If the statement is too complex to process in this way (possibly due to the setting of database options or characteristics of the schema or query), then the statement fails and the database server returns an error.

option-name = option-value

Specify an option setting. The setting you specify is only applicable to the current statement and takes precedence over any public or temporary option settings, including those set by ODBC-enabled applications.

The supported options are:

- isolation_level option
- max_query_tasks option
- optimization_goal option
- optimization_level option
- optimization_workload option
- user_estimates option
- parameterization_level

If you specify the isolation_level option in a query, the value specified in the query takes precedence over all other isolation level settings (such as setting the isolation_level option for the database or the setting for the cursor) for the current query.

sequence-expression

To return the next value in the sequence, use `[owner.]sequence-name.NEXTVAL`. The sequence is shared by all connections, so each connection will get a unique next value.

To return the most recently supplied sequence value for the current connection, use `[owner.]sequence-name.CURRVAL`.

Remarks

The SELECT statement can be used:

- for retrieving results from the database.
- in Interactive SQL to browse data in the database, or to export data from the database to an external file.
- in procedures and triggers or in Embedded SQL. A SELECT statement with an INTO clause is used for retrieving results from the database when the SELECT statement only returns one row. For multiple row queries, you must use cursors.
- to return a result set from a procedure.

Privileges

You must have the appropriate SELECT privileges on the objects referred to in the SELECT statement.

To select the CURRVAL or NEXTVAL values from a sequence generator, you must have USE ANY SEQUENCE system privilege, or be the owner of the sequence, or have been granted the required privileges to use the sequence generator.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Core feature. Check individual clauses against the standard. For example, the ROLLUP keyword, which can be specified in a GROUP BY clause, is part of optional ANSI/ISO SQL Language Feature T431. Some of the optional ANSI/ISO SQL Language Features supported by the software include:

- The WINDOW clause and WINDOW aggregate functions comprise optional ANSI/ISO SQL Language Features T611 and T612.
- Sequence expressions are part of ANSI/ISO SQL Feature T176.
- Common table expressions are optional ANSI/ISO SQL Language Feature T121. A common table expression included in a nested query expression is Feature T122. WITH RECURSIVE is optional ANSI/ISO SQL Language Feature T131; if included in a nested query it constitutes Feature T132.
- The ability to specify an ORDER BY clause with a query expression involving UNION, EXCEPT, or INTERSECT is optional ANSI/ISO SQL Feature F850. The ability to specify ORDER BY in a subquery is ANSI/ISO SQL Feature F851.
- In the ANSI/ISO SQL Standard, FOR UPDATE and FOR READ ONLY are part of a cursor declaration.

The software offers support for many extensions to the ANSI/ISO SQL definition of the SELECT statement. Some of these include:

- The optional `cursor-concurrency` clause (FOR UPDATE BY { LOCK | VALUES | TIMESTAMP}) is not part of the ANSI/ISO SQL Standard.
- The FOR XML, OPTION, and INTO clauses are not part of the ANSI/ISO SQL Standard.
- The row limitation clause is not part of the ANSI/ISO SQL Standard. In the ANSI/ISO SQL Standard, row limitation is supported using FETCH FIRST syntax, which is optional Language Feature F856. The syntax for Feature F856 is not supported by the software.
- The ability to specify ORDER BY `n` is not part of the ANSI/ISO SQL Standard.
- In the ANSI/ISO SQL Standard, all cursors except INSENSITIVE cursors are updatable by default. The read-only default with Embedded SQL programs is not part of the ANSI/ISO SQL Standard.

Transact-SQL

There are substantial differences in SELECT statement support between SQL Anywhere and Adaptive Server Enterprise. Several features of the SELECT statement are not supported by Adaptive Server Enterprise.

These differences include:

- SAP ASE does not support SQL Anywhere's cursor concurrency clause; to acquire a lock on a fetched row, you must use the HOLDLOCK table hint.
- Adaptive Server Enterprise does not support recursive queries or common table expressions.
- There are differences between Adaptive Server Enterprise and SQL Anywhere with respect to Transact-SQL outer joins.

In Transact-SQL you use the SELECT statement to assign a value to a variable, rather than with the Watcom SQL SET statement.

Example

This query returns the total number of employees in the Employees table.

```
SELECT COUNT(*)
FROM GROUPO.Employees;
```

This query lists all customers and the total value of their orders.

```
SELECT CompanyName,
   CAST( SUM( SalesOrderItems.Quantity *
   Products.UnitPrice ) AS INTEGER ) VALUE
FROM GROUPO.Customers
   JOIN GROUPO.SalesOrders
   JOIN GROUPO.SalesOrderItems
   JOIN GROUPO.Products
GROUP BY CompanyName
ORDER BY VALUE DESC;
```

The following statement shows an Embedded SQL SELECT statement where the number of employees in the Employees table is selected into the :size host variable:

```
SELECT COUNT(*) INTO :size
FROM GROUPO.Employees;
```

The following statement is optimized to return the first row in the result set quickly:

```
SELECT Name
FROM GROUPO.Products
GROUP BY Name
HAVING COUNT( * ) > 1
AND MAX( UnitPrice ) > 10
OPTION( optimization_goal = 'first-row' );
```

The following statement creates the function GetCustomer that declares a row variable and executes a query that uses the INTO VARIABLE clause to return the updated row value:

```
CREATE FUNCTION GetCustomer ( @custid Customers.ID%TYPE )
RETURNS Customer%ROWTYPE
BEGIN
   DECLARE @customer Customers%ROWTYPE;
```

```
    SELECT * INTO VARIABLE @customer
    FROM Customers
    WHERE id = @custid;
    RETURN @customer;
END;
```

Related Information

Advanced: Query Processing Phases
Eligibility to Skip Query Processing Phases
Row Limitation Clauses in SELECT, UPDATE, and DELETE Query Blocks
Joins: Retrieving Data from Several Tables
Common Table Expressions
Use of a Sequence to Generate Unique Values
Use of the FOR XML Clause to Retrieve Query Results as XML
Use of the FOR JSON Clause to Retrieve Query Results as JSON
ODBC Cursor Characteristics
Updatable Statements
Cursor Attributes
Transact-SQL Outer Joins (*= or =*)
Transact-SQL Query Support
Expressions in SQL Statements [page 34]
FROM Clause [page 1173]
GROUP BY Clause [page 1215]
Search Conditions [page 58]
WINDOW Clause [page 1489]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
UNION Statement [page 1450]
EXCEPT Statement [page 1148]
INTERSECT Statement [page 1242]
materialized_view_optimization Option
ansi_update_constraints Option
reserved_keywords Option
Column '%1' not found
FOR XML EXPLICIT

1.4.4.246 SET Statement

Assigns a value to a SQL variable.

Syntax

```
SET [ owner.]identifier = expression
```

Remarks

The SET statement assigns a new value to a variable. The variable must have been previously created by using a CREATE VARIABLE statement or DECLARE statement, or it must be an OUTPUT parameter for a procedure. The variable name can optionally use the Transact-SQL convention of an @ sign preceding the name. For example: `SET @localvar = 42`.

A variable can be used in a SQL statement anywhere a column name is allowed. If a column name exists with the same name as the variable, then the column value is used.

The `owner` specification is only for use when setting owned database-scope variables.

Variables are necessary for creating large text or binary objects for INSERT or UPDATE statements from Embedded SQL programs because Embedded SQL host variables are limited to 32767 bytes.

Variables are local to the current connection and disappear when you disconnect from the database or use the DROP VARIABLE statement. They are not affected by COMMIT or ROLLBACK statements.

If you set a database-scope variable, however, the variable persists after a disconnect. When the database is restarted, the value of a database-scope variable reverts to NULL or its default, if defined. The SYSDATABASEVARIABLE system view contains a list of all database-scope variables and their initial values.

You cannot set a database-scope variable owned by another user.

Privileges

No privileges are required to set a self-owned database-scope variable. To set a database-scope variable owned by PUBLIC, you must have the UPDATE PUBLIC DATABASE VARIABLE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Part of optional Language Feature P002, "Computational completeness".

Transact-SQL

The SET statement is supported by SAP Adaptive Server Enterprise. In Adaptive Server Enterprise, a single SET statement can be used to assign values to multiple variables, with individual assignment clauses separated by commas.

Example

This simple example shows the creation of a variable called birthday, and sets the date to CURRENT DATE.

```
CREATE VARIABLE @birthday DATE;
SET @birthday = CURRENT DATE;
```

The following code fragment inserts a large text value into the database.

```
size_t  size;
FILE *  fp;
EXEC SQL BEGIN DECLARE SECTION;
DECL_VARCHAR( 5000 ) buffer;
EXEC SQL END DECLARE SECTION;
fp = fopen( "blob.dat", "r" );
EXEC SQL CREATE VARIABLE hold_blob LONG VARCHAR;
EXEC SQL SET hold_blob = '';
for(;;) {
    size = fread( (void *)buffer.array, 1, 5000, fp );
    if( size <= 0 ) break;
    buffer.len = (a_sql_ulen) size;
    EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES( 1, hold_blob );
EXEC SQL COMMIT;
EXEC SQL DROP VARIABLE hold_blob;
fclose( fp );
```

Related Information

Embedded SQL Host Variable Usage
SQL Variables [page 123]
CREATE VARIABLE Statement [page 1047]
DECLARE Statement [page 1057]
SET Statement [T-SQL] [page 1375]
DROP VARIABLE Statement [page 1145]
Expressions in SQL Statements [page 34]

1.4.4.247 SET Statement [T-SQL]

Sets database options for the current connection in an Adaptive Server Enterprise-compatible manner.

Syntax

```
SET option-name option-value
```

Remarks

The available options are as follows:

Option name	Option value
ansinull	On or Off
ansi_permissions	On or Off
close_on_endtrans	On or Off
datefirst	1, 2, 3, 4, 5, 6, or 7 The setting of this option affects the DATEPART function when obtaining a weekday value.
quoted_identifier	On | Off
rowcount	`integer`
self_recursion	On | Off
string_rtruncation	On | Off
textsize	`integer`
transaction isolation level	0, 1, 2, 3, snapshot, statement snapshot, or read only statement snapshot

Database options in SQL Anywhere are set using the SET OPTION statement. However, SQL Anywhere also provides support for the Adaptive Server Enterprise SET statement for options that are useful for compatibility.

The following options can be set using the Transact-SQL SET statement in SQL Anywhere and Adaptive Server Enterprise:

SET ansinull

The default behavior for comparing values to NULL is different in SQL Anywhere and Adaptive Server Enterprise. Setting ansinull to Off provides Transact-SQL compatible comparisons with NULL.

SQL Anywhere also supports the following syntax:

```
SET ansi_nulls { On | Off }
```

SET ansi_permissions

The default behavior is different in SQL Anywhere and Adaptive Server Enterprise regarding privileges required to perform an UPDATE or DELETE containing a column reference. Setting ansi_permissions to Off provides Transact-SQL-compatible privileges on UPDATE and DELETE.

SET close_on_endtrans

The default behavior is different in SQL Anywhere and Adaptive Server Enterprise for closing cursors at the end of a transaction. Setting close_on_endtrans to Off provides Transact-SQL compatible behavior.

SET datefirst

The default is 7, which means that the first day of the week is by default Sunday.

SET quoted_identifier

Controls whether strings enclosed in double quotes are interpreted as identifiers (On) or as literal strings (Off).

SET rowcount

`integer` The Transact-SQL ROWCOUNT option limits the number of rows fetched for any cursor to the specified integer. This includes rows fetched by re-positioning the cursor. Any fetches beyond this maximum return a warning. The option setting is considered when returning the estimate of the number of rows for a cursor on an OPEN request.

SET ROWCOUNT also limits the number of rows affected by a searched UPDATE or DELETE statement to `integer`. This might be used, for example, to allow COMMIT statements to be performed at regular intervals to limit the size of the rollback log and lock table. The application (or procedure) would need to provide a loop to cause the update/delete to be re-issued for rows that are not affected by the first operation. A simple example is given below:

```
BEGIN
   DECLARE @count INTEGER
   SET rowcount 20
   WHILE(1=1) BEGIN
      UPDATE GROUPO.Employees SET Surname='new_name'
      WHERE Surname <> 'old_name'
      /* Stop when no rows changed */
      SELECT @count = @@rowcount
      IF @count = 0 BREAK
      PRINT string('Updated ',
               @count,' rows; repeating...')
      COMMIT
   END
   SET rowcount 0
END
```

If the ROWCOUNT setting is greater than the number of rows that Interactive SQL can display, Interactive SQL may do some extra fetches to reposition the cursor. So, the number of rows actually displayed may be less than the number requested. Also, if any rows are re-fetched due to truncation warnings, the count may be inaccurate.

A value of zero resets the option to get all rows.

SET self_recursion

The self_recursion option is used within triggers to enable (On) or prevent (Off) operations on the table associated with the trigger from firing other triggers.

SET string_rtruncation

The default behavior is different between SQL Anywhere and Adaptive Server Enterprise when non-space characters are truncated during assignment of SQL string data. Setting string_rtruncation to On provides Transact-SQL-compatible string comparisons.

SET textsize

Specifies the maximum size (in bytes) of TEXT or IMAGE type data to be returned with a SELECT statement. The @@textsize global variable stores the current setting. To reset to the default size (32 KB), use the command:

SET transaction isolation level

Sets the locking isolation level for the current connection.

For Adaptive Server Enterprise, only 1 and 3 are valid options. For SQL Anywhere, any of 0, 1, 2, 3, snapshot, statement snapshot, and read only statement snapshot is a valid option.

The SET statement is allowed by SQL Anywhere for the prefetch option, for compatibility, but has no effect.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Related Information

Isolation Levels and Consistency
Options for Transact-SQL Compatibility
Transact-SQL and ANSI/ISO SQL Standard Compatibility Options
ansinull Option
isolation_level Option
ansi_permissions Option
close_on_endtrans Option
quoted_identifier Option
string_rtruncation Option
first_day_of_week Option
DATEPART Function [Date and Time] [page 322]
SET OPTION Statement [page 1387]

1.4.4.248 SET CONNECTION Statement [Interactive SQL] [ESQL]

Changes the active database connection.

Syntax

```
SET CONNECTION [ connection-name ]
```

```
connection-name :
identifier
```

```
| string
| hostvar
```

Remarks

The SET CONNECTION statement changes the active database connection to connection-name. The current connection state is saved, and is resumed when it again becomes the active connection. If connection-name is omitted and there is a connection that was not named, that connection becomes the active connection.

When cursors are opened in Embedded SQL, they are associated with the current connection. When the connection is changed, the cursor names of the previously active connection become inaccessible. These cursors remain active and in position, and become accessible when the associated connection becomes active again.

This SQL statement is not supported for SAP HANA databases.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

SET CONNECTION is part of optional ANSI/ISO SQL Language Feature F771, "Connection management". Its usage within an Interactive SQL session is not in the standard.

Example

The following example fragment is Embedded SQL.

```
EXEC SQL SET CONNECTION :conn_name;
```

From Interactive SQL, set the current connection to the fictitious connection conn1.

```
SET CONNECTION conn1;
```

Related Information

Interactive SQL
CONNECT Statement [ESQL] [Interactive SQL] [page 815]
DISCONNECT Statement [ESQL] [Interactive SQL] [page 1085]

1.4.4.249 SET DESCRIPTOR Statement [ESQL]

Describes the variables in a SQL descriptor area and to place data into the descriptor area.

Syntax

```
SET DESCRIPTOR descriptor-name
{ COUNT = { integer | hostvar }
| VALUE { integer | hostvar } assignment, ... }
```

```
assignment :
{ TYPE | SCALE | PRECISION | LENGTH | INDICATOR }
      = { integer | hostvar }
| DATA = hostvar
```

```
descriptor-name : identifier
```

Remarks

The SET DESCRIPTOR statement is used to describe the variables in a descriptor area, and to place data into the descriptor area.

The SET...COUNT statement sets the number of described variables within the descriptor area. The value for count must not exceed the number of variables specified when the descriptor area was allocated.

The value { `integer` | `hostvar` } specifies the variable in the descriptor area upon which the assignment(s) is performed.

Type checking is performed when doing SET...DATA, to ensure that the variable in the descriptor area has the same type as the host variable. LONG VARCHAR and LONG BINARY are not supported by SET DESCRIPTOR...DATA.

If an error occurs, the code is returned in the SQLCA.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

SET DESCRIPTOR is part of optional ANSI/ISO SQL Language feature B031, "Basic dynamic SQL".

Example

The following example sets the type of the column with position col_num in sqlda.

```
void set_type( SQLDA *sqlda, int col_num, int new_type )
{
    EXEC SQL BEGIN DECLARE SECTION;
    INT new_type1 = new_type;
    INT col = col_num;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SET DESCRIPTOR sqlda VALUE :col TYPE = :new_type1;
}
```

Related Information

The SQL Descriptor Area (SQLDA)
ALLOCATE DESCRIPTOR Statement [ESQL] [page 660]
DEALLOCATE DESCRIPTOR Statement [ESQL] [page 1054]

1.4.4.250 SET MIRROR OPTION Statement

Changes the values of options that control the settings for database mirroring and read-only scale-out.

Syntax

```
SET MIRROR OPTION option-name={ option-value | NULL }
```

```
option-name :
authentication_string
auto_add_fan_out
auto_add_server
auto_failover
child_creation
```

max_disconnected_time
max_logfile_size
max_retry_connect_time
page_timeout
promotion_time
synchronization_mode
lagtime

Parameters

NULL

Specifies the default value for the option. When the `option-name` is set to NULL, the option value is set to its default value.

`option-name`	Applies to	Values	Default	Description
authentication_string	database mirroring	string	null	Specifies the authentication string used by all the servers in the database mirroring system. The authentication string is required for database mirroring.
auto_add_fan_out	read-only scale-out	integer	10	Specifies the maximum number of children for each branch. The minimum value that can be specified is 2.
auto_add_server	read-only scale-out	string	null	Specifies the name of the database server that acts as the parent of the automatic assignment tree.

option-name	Applies to	Values	Default	Description
auto_failover	database mirroring	on, off	null	Specifies whether the mirror server automatically takes over as the primary server when the current primary server goes down. This option does not apply to synchronous mode. This option accepts Boolean values (automatic failover is turned on with YES, ON, TRUE, or 1, and is turned off with any of NO, OFF, FALSE, and 0). The parameters are case insensitive. If you are using asynchronous or asyncfullpage mode, set the auto_failover option to on. Then, if the primary server goes down, the mirror server automatically takes over as the primary server.
child_creation	read-only scale-out	automatic, off, manual	automatic	Controls whether copy nodes are created automatically.
max_disconnected_time	read-only scale-out	integer, in seconds, greater than or equal to max_retry_connect_time	no time limit	Specifies the amount of time since the last time the copy node was connected to the parent, alternate parent, or root database before the copy node shuts down.

option-name	Applies to	Values	Default	Description
max_logfile_size	database mirroring and read-only scale-out	integer, with optional K, M, G suffix	no maximum size limit	Specifies a maximum size, in bytes, for mirror debug log files. Use K, M, or G to specify kilobytes, megabytes, or gigabytes. The minimum size is 10 KB. Once the maximum size is reached, the log file is renamed with the extension `.old` and a new file is created. Specify a mirror debug log file by setting the *logfile* `server-option` in the CREATE MIRROR SERVER or ALTER MIRROR SERVER statement.
max_retry_connect_time	read-only scale-out	integer, in seconds	120	Specifies the length of time that a copy node attempts to reconnect to its parent once the parent becomes unavailable.
page_timeout	database mirroring	integer, in seconds	5	Specifies how often, in seconds, transaction log pages are sent to the mirror server, whether or not they are full. This option applies only when using asyncfullpage mode.

option-name	Applies to	Values	Default	Description
promotion_time	read-only scale-out	integer, in seconds, greater than or equal to max_retry_connect_time	3600	Specifies the length of time that a copy node stays connected to the root database server after a parent connection is lost before promoting itself (adjusting the scale-out tree to not have a disconnected parent). This option helps to avoid adjusting the scale-out tree if a copy node is down for a short time (which can result in a shallow scale-out tree), while attempting to avoid increased load on the primary database for long periods of time. To never promote, set this option to 315,360,000 (10 years) or higher. This option is only supported as of version 16.
synchronization_mode	database mirroring	synchronous, asynchronous, asyncfullpage	synchronous	Specifies the synchronization mode used for database mirroring: synchronous (sync), asynchronous (async), or asyncfullpage (page). The synchronization mode controls when and how transactions are recorded on the mirror server.

option-name	Applies to	Values	Default	Description
lagtime	database mirroring	integer, in seconds	60	Specifies the approximate length of time the mirror server may run behind the primary server in applying the transaction log. The primary server reduces its rate of transactions when the lag time approaches the value of this setting. Set the value to 0 or any integer between 15 and 3600 seconds (one hour). Setting the option to 0 is equivalent to unlimited time.

Remarks

Once you create a database server for a database mirroring system or a read-only scale-out system by using the CREATE MIRROR SERVER statement, use the SET MIRROR OPTION statement to configure the settings for the system.

Read-only scale-out and database mirroring each require a separate license.

Privileges

You must have started the server, or have the MANAGE ANY MIRROR SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement sets the authentication string for a database mirroring system to abc:

```
SET MIRROR OPTION authentication_string = 'abc';
```

Related Information

Automatically Assign the Parent of a Copy Node
How Read-only Scale-out Systems Handle the Loss of a Parent Connection
Database Mirroring Modes
How Child Copy Nodes Are Added
Database Mirroring
Read-only Scale-out
Separately Licensed Components
Tip: Check the Lagtime Mirror Option Setting (Mirrored Systems Only)
-xa Database Server Option
CREATE MIRROR SERVER Statement [page 901]
ALTER MIRROR SERVER Statement [page 695]
DROP MIRROR SERVER Statement [page 1105]
SYSMIRROROPTION System View [page 1923]

1.4.4.251 SET OPTION Statement

Changes the values of database and connection options.

Syntax

Specify an option

```
SET [ EXISTING ] [ TEMPORARY ] OPTION
  [ userid.| PUBLIC.]option-name = [ option-value ]
```

Specify an identifier (deprecated)

```
SET [ EXISTING ] [ TEMPORARY ] OPTION
  [ userid.| PUBLIC.]option-name = [ identifier ]
```

```
userid : identifier
```

```
option-name : identifier
```

```
option-value : ON, OFF, NULL, string literal, number, hostvar, or
@variable-name
```

Parameters

option-value

When using the syntax for specifying an option, `option-value` can be one of:

- the keywords ON, OFF, or NULL
- a string literal value, within single quotation marks
- a number of any valid format, including NUMERIC
- within an Embedded SQL program, the value of a host variable hostvar
- the value of a SQL variable with a variable name that must begin with an @ sign

When using the syntax for specifying an identifier, you can specify any valid identifier as an option value. Also, the database server treats the name of the identifier as if it were a string literal enclosed within single quotes. For example, the following two statements are equivalent:

```
SET TEMPORARY OPTION ansi_update_constraints = 'strict';
```

```
SET TEMPORARY OPTION ansi_update_constraints = strict;
```

Remarks

The SET OPTION statement is used to change options that affect the behavior of the database server. Setting the value of an option can change the behavior for all users (public), for an individual user (including user-extended roles), or for the current connection. The new setting can be made either temporary or permanent.

Setting an option for a role is only meaningful for user-extended roles, and only applies to that user when they are logged in. A user who inherits a role does not inherit option settings.

The classes of options that can be set with the SET OPTION statement are:

- Transact-SQL compatibility options
- connection and database options
- synchronization options
- user-defined options

Option scope

With most options, you can set their value at three levels of scope: public, user, and connection. Some specific options, such as login_mode, are restricted to the public level only. A connection option takes precedence over the other two levels, and user options take precedence over public options. You set a connection-level option by using the TEMPORARY keyword. If you set a user-level option for the current user, the corresponding connection-level option is set at the same time.

By default, the option value applies to the currently logged on user ID that executed the SET OPTION statement. If you specify a user ID, the option value applies to that user. If you specify PUBLIC, the option value applies to all users who do not have an individual setting for the option.

TEMPORARY options

By default, a new option value is made permanent unless the TEMPORARY keyword is specified. Adding the TEMPORARY keyword to the SET OPTION statement affects the duration of the change.

When the SET TEMPORARY OPTION statement is not qualified with a user ID, the new option value is in effect only for the current connection.

When SET TEMPORARY OPTION is used for the PUBLIC role, the change is in place for as long as the database is running. When the database is shut down, TEMPORARY options for the PUBLIC role revert back to their permanent value.

Setting temporary options for the PUBLIC role offers a security benefit. For example, when the login_mode option is enabled, the database relies on the login security of the system on which it is running. Enabling it temporarily means that a database relying on the security of a Windows domain is not compromised if the database is shut down and copied to a local computer. In that case, the temporary enabling of the login_mode option reverts to its permanent value, which could be Standard, a mode where integrated logins are not permitted.

Removing option settings

If `option-value` is omitted, the specified option setting is deleted from the database. If it was a user-level option setting, the value reverts back to the PUBLIC setting. If a TEMPORARY option is deleted, the option setting reverts back to the permanent setting for that user.

Option data types

Options can have Boolean, numeric, or string values, but are always stored as strings in the database. Option settings are always returned as strings as the result of a property function or when returned as a result of a function or system stored procedure. Option values cannot be larger than the database page size.

User-defined options

Any option, whether user-defined or not, must have a public setting before a user-specific value can be assigned. The database server does not support setting TEMPORARY values for user-defined options. For example, to create a user-defined option named ApplicationControl, you first execute the statement:

```
SET OPTION PUBLIC.ApplicationControl = 'Default';
```

This statement sets the ApplicationControl option to Default for all users, and takes effect with each new connection to the server. Subsequently, an individual user may establish their own setting for this option by executing a separate SET OPTION statement.

Restrictions

If you use the EXISTING keyword, option values cannot be set for an individual user ID unless there is already a PUBLIC setting for that option.

> ⚠ Caution
>
> Do not change option values while a cursor is open. Changing the option values while a cursor is open can lead to inconsistent results within the cursor. For example, changing the date_format option while a cursor is open can result in some rows being returned in the old format and some rows returned in the new format. To ensure that the rows in the result set are computed consistently using the new option value, open the cursor after the option value is changed.

There are several ways you can query the value of specific options for a connection or user.

The SET OPTION statement is ignored by the SQL Flagger.

Privileges

Any user can set their own options.

To set database options for other users or roles, including the PUBLIC role, you must have one of the following system privileges, depending upon which privilege the option requires:

- SET ANY SYSTEM OPTION
- SET ANY PUBLIC OPTION
- SET ANY SECURITY OPTION
- SET ANY USER DEFINED OPTION

Side Effects

Unless TEMPORARY is specified, an automatic commit is performed.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Set the date format option for all users without an individual setting:

```
SET OPTION PUBLIC.date_format = 'Mmm dd yyyy';
```

Set the wait_for_commit option to On:

```
SET OPTION wait_for_commit = 'On';
```

The following fragment is an Embedded SQL example:

```
EXEC SQL SET TEMPORARY OPTION date_format = :value;
```

Set the date_format option for the user that is currently connected. Future connections for the same user ID use this option value.

```
SET OPTION date_format = 'yyyy/mm/dd';
```

The following statement removes the setting of the date_format option for the current user ID. After executing this statement, the date_format setting for PUBLIC is used instead.

```
SET OPTION date_format=;
```

The following statement changes the login_mode option to Standard for all users.

```
SET OPTION PUBLIC.login_mode = 'Standard';
```

Related Information

Database Options
Alphabetical List of Database Options
How to View Database Options
Remote ID Settings
Transact-SQL and ANSI/ISO SQL Standard Compatibility Options
Synchronization Options
SQL Remote Options
SYSOPTION System View [page 1929]
sa_conn_options System Procedure [page 1553]
sa_conn_options System Procedure [page 1553]
CONNECTION_PROPERTY Function [System] [page 293]
GET OPTION Statement [ESQL] [page 1189]
SET OPTION Statement [Interactive SQL] [page 1391]
SET Statement [T-SQL] [page 1375]
SET REMOTE OPTION Statement [SQL Remote] [page 1394]

1.4.4.252 SET OPTION Statement [Interactive SQL]

Changes the values of Interactive SQL options.

Syntax

Set an Interactive SQL option

```
SET OPTION option-name = [ option-value ] | SET TEMPORARY OPTION option-name = [ option-value ]
```

```
option-name : identifier | string | hostvar
```

```
option-value : string | identifier | number
```

Save current Interactive SQL options permanently

```
SET PERMANENT
```

List current database option settings

```
SET
```

Remarks

When you set an option using the SET OPTION syntax, the option setting is stored permanently and does not change unless another SET OPTION statement changes it.

Using the SET TEMPORARY OPTION syntax allows you to temporarily change an option setting. The temporary setting remains in effect until you close Interactive SQL. The next time you start Interactive SQL, the option reverts to its permanent setting.

Use the SET PERMANENT syntax to permanently save all current Interactive SQL option settings (any temporary settings become permanent).

If `option-value` is omitted, the specified option is set to its default value.

When you list the current database option settings, if there are temporary options settings for the database server, they are displayed instead of the permanent settings.

Interactive SQL option settings are stored on the client computer, not in the database.

The following table lists the Interactive SQL options.

Option	Values	Default
auto_commit option [Interactive SQL]	On, Off	Off
auto_refetch option [Interactive SQL]	On, Off	On
bell option [Interactive SQL]	On, Off	On
command_delimiter option [Interactive SQL]	String	' ; '
commit_on_exit option [Interactive SQL]	On, Off	On
default_isql_encoding option [Interactive SQL]	String	Empty string
echo option [Interactive SQL]	On, Off	On
input_format option [Interactive SQL]	TEXT, EXCEL, FIXED, SHAPEFILE	TEXT
isql_allow_read_client_file option [Interactive SQL]	On, Off, Prompt	Prompt
isql_allow_write_client_file option [Interactive SQL]	On, Off, Prompt	Prompt
isql_command_timing option [Interactive SQL]	On, Off	On
isql_escape_character option [Interactive SQL]	Character	' \ '
isql_field_separator option [Interactive SQL]	String	' , '
isql_maximum_displayed_rows option [Interactive SQL]	All or a non-negative integer	500
isql_print_result_set option [Interactive SQL]	Last, All, None	Last

Option	Values	Default
isql_quote option [Interactive SQL]	String	'
nulls option [Interactive SQL]	String	'(NULL)'
on_error option [Interactive SQL]	Stop, Continue, Prompt, Exit, Notify_Continue, Notify_Stop, Notify_Exit	Prompt
output_format option [Interactive SQL]	TEXT, EXCEL, FIXED, HTML, SQL, XML	TEXT
output_length option [Interactive SQL]	Integer	0
output_nulls option [Interactive SQL]	String	Empty string
truncation_length option [Interactive SQL]	Integer	256

Privileges

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example changes the value of the on_error option to continue:

```
SET OPTION on_error='continue';
```

Related Information

Interactive SQL Options
Interactive SQL
auto_commit Option [Interactive SQL]
auto_refetch Option [Interactive SQL]
bell Option [Interactive SQL]
command_delimiter Option [Interactive SQL]
commit_on_exit Option [Interactive SQL]
default_isql_encoding Option [Interactive SQL]

1.4.4.253 SET REMOTE OPTION Statement [SQL Remote]

Sets a message control parameter for a SQL Remote message system.

Syntax

```
SET REMOTE message-system OPTION
[ userid.| PUBLIC.]option-name = option-value
```

```
message-system :
FILE
| FTP
| HTTP
| SMTP
```

```
option-name :
common-options
| file-options
| ftp-options
| smtp-options
```

```
common-options :
debug
| encode_dll
| max_retries
| output_log_send_on_error
| output_log_send_limit
| output_log_send_now
| pause_after_failure
```

```
file-options :
directory
| invalid_extensions
| unlink_delay
```

```
ftp-options :
```

```
active_mode
| host
| invalid_extensions
| password
| port
| root_directory
| reconnect_retries
| reconnect_pause
| suppress_dialogs
| user

http-options :
| certificate
| client_port
| https
| password
| proxy
| reconnect_retries
| reconnect_pause
| root_directory
| url
| user

smtp-options :
 local_host
| pop3_host
| pop3_password
| pop3_port
| pop3_userid
| smtp_authenticate
| smtp_option
| smtp_password
| smtp_port
| smtp_userid
| suppress_dialogs
| top_supported

option-value : string
```

Parameters

userid

If you do not specify a `userid`, then the current publisher is assumed.

common-options

These options are common to the FILE, FTP, HTTP, and SMTP message systems:

debug

This parameter is set either to YES or NO. The default is NO. When set to YES, debug output specific to the message system is displayed. This information can be used for troubleshooting problems in the message system.

max_retries

By default, when SQL Remote is running in continuous mode and an error occurs when accessing the message system, it shuts down after the send and/or received phases. Use this parameter to specify the number of times you want SQL Remote to retry the send and/or receive phases before it shuts down.

output_log_send_on_error

Sends log information when an error occurs.

output_log_send_limit

Limits the amount of information that is sent to the consolidated database. The output_log_send_limit option specifies the number of bytes at the end of the output log (that is, the most recent entries) that are sent to the consolidated database. The default is 5K.

output_log_send_now

When set to YES, sends output log information to the consolidated database. On the next poll, the remote database sends the output log information and then resets the output_log_send_now option to NO.

pause_after_failure

This parameter applies when the max_retries parameter is specified to a value other than zero and SQL Remote is running in continuous mode. When an error occurs in the message system, this parameter defines the number of seconds SQL Remote waits between retrying the send and/or receive phases.

encode_dll

If you have implemented a custom encoding scheme, you must set this to the full path of the custom encoding DLL that you created.

file-options

These options apply to the FILE message system only:

directory

The directory under which the messages are stored. This parameter is an alternative to the SQLREMOTE environment variable.

invalid_extensions

A comma-separated list of file extensions that you do not want the SQL Remote Message Agent (dbremote) to use when generating files in the messaging system.

unlink_delay

The number of seconds to wait before attempting to delete a file if the previous attempt to delete the file failed. If no value is defined for unlink_delay, then the default behavior is set to pause for 1 second after the first failed attempt, 2 seconds after the second failed attempt, 3 seconds after the third failed attempt, and 4 seconds after the fourth failed attempt.

ftp-options

These options apply to the FTP message system only:

active_mode

This parameter controls how SQL Remote establishes the server/client connection. This parameter is set either to YES or NO. The default is NO (passive mode). Passive mode is the preferred transfer mode and the default for the FTP message system. In passive mode, all data transfer connections are

initiated by the client, in this case, the message system. In active mode, the FTP server initiates all data connections.

host

The host name of the computer where the FTP server is running. This parameter can be a host name (such as ftp.sap.com) or an IP address (such as 192.138.151.66).

invalid_extensions

A comma-separated list of file extensions that you do not want dbremote to use when generating files in the messaging system.

password

The password for accessing the FTP host.

port

The IP port number used for the FTP connection. This parameter is usually not required.

reconnect_retries

The number of times the message system should try to open a socket with the server before failing. The default value is 4. When you set this parameter, only reconnections are affected. The initial connection made by the FTP message system is not affected.

reconnect_pause

The time in seconds to pause between each connection attempt. The default setting is 30 seconds. When you set this parameter, only reconnections are affected. The initial connection made by the FTP message system is not affected.

root_directory

The root directory within the FTP host site that the messages are stored under.

suppress_dialogs

This parameter is set to ON or OFF. The default value is OFF. If it is set to ON, the *Connect* window does not appear after failed attempts to connect to the messaging system. Instead, an error is generated.

user

The user name for accessing the FTP host.

http-options

These options apply to the HTTP message system only:

certificate

To make a secure (HTTPS) request, a client must have access to the certificate used by the HTTPS server. The necessary information is specified in a string of semicolon-separated keyword=value pairs. You can use the file keyword to specify the file name of the certificate. You cannot specify a file and certificate keyword together. The following keywords are available:

Keyword	Abbreviation	Description
file		The file name of the certificate
certificate	cert	The certificate itself
company	co	The company specified in the certificate

Keyword	Abbreviation	Description
unit		The company unit specified in the certificate
name		The common name specified in the certificate

Certificates are required only for requests that are either directed to an HTTPS server or can be redirected from a non-secure to a secure server. Only PEM formatted certificates are supported (for example, certificate='file=filename'

To create a certificate name in a SQL Anywhere database:

```
CREATE OR REPLACE CERTIFICATE certificate_name FROM FILE
'certificate_file';
```

To use the certificate name for an HTTPS message type:

```
SET REMOTE HTTP OPTION user_name.certificate= 'cert_name=certificate_name';
```

client_port

Identifies the port number on which SQL Remote communicates using HTTP. It is provided for, and recommended only for, connections through firewalls that filter "outgoing" TCP/IP connections. You can specify a single port number, ranges of port numbers, or a combination of the two. Specifying a low number of client ports could result in SQL Remote being unable to send and receive messages if the operating system has not released the ports in a timely manner after SQL Remote closes the port on a previous run.

debug

When set to YES, all HTTP commands and responses are displayed in the output log. This information can be used for troubleshooting HTTP support problems. The default is NO.

https

Specify whether to use HTTPS (**https=yes**) or HTTP (**https=no**).

password

The message server database password. The password authenticates to third-party HTTP servers and gateways using RFC 2617 Basic authentication.

proxy_host

Specifies the URI of a proxy server. For use when SQL Remote must access the network through a proxy server. Indicates that SQL Remote is to connect to the proxy server and send the request to the message server through it.

reconnect_retries

The number of times the message system should try to open a socket with the server before failing. The default value is 4. When you set this parameter, only reconnections are affected. The initial connection made by the FTP message system is not affected.

reconnect_pause

The time in seconds to pause between each connection attempt. The default setting is 30 seconds. When you set this parameter, only reconnections are affected. The initial connection made by the FTP message system is not affected.

root_directory

This HTTP control parameter is ignored when specified at the client side. You define this control parameter in the message server prior to calling the sr_add_message_server or sr_update_message_server stored procedure. When using the HTTP message system, the address specified for a remote user or publisher can only contain a single subdirectory, and not multiple subdirectories.

url

Specify the server name or IP address and optionally the port number of the HTTP server being used, separated by a semicolon. If requests are being passed through the Relay Server, you can optionally add a URL extension to indicate which server farm the request should be passed to.

user

The message server database user ID. Authenticates to third-party HTTP servers and gateways using RFC 2617 Basic authentication.

smtp-options

These options apply to the SMTP message system only:

local_host

The name of the local computer. It is useful on computers where SQL Remote is unable to determine the local host name. The local host name is needed to initiate a session with any SMTP server. In most network environments, the local host name can be determined automatically and this entry is not needed.

pop3_host

The name of the computer on which the POP host is running. Typically, it is the same name as the SMTP host. It corresponds to the POP3 host field in the SMTP/POP3 login window.

pop3_password

The password used to retrieve mail. It corresponds to the password field in the SMTP/POP3 login window.

pop3_port

The number of the port on which the POP server is listening. The default is 110. This corresponds to the port field in the SMTP/POP3 login window.

pop3_userid

The user ID used to retrieve mail. The POP user ID corresponds to the user ID field in the SMTP/POP3 login window. You must obtain a user ID from your POP host administrator.

smtp_host

The name of the computer on which the SMTP server is running. It corresponds to the SMTP host field in the SMTP/POP3 login window

top_supported

SQL Remote uses a POP3 command called TOP when enumerating incoming messages. The TOP command may not be supported by all POP servers. When you set the top_supported parameter to

NO, SQL Remote uses the RETR command, which is less efficient but works with all POP servers. The default is YES.

smtp_authenticate

Determines whether the SMTP message system authenticates the user. The default value is YES. Set this parameter to NO to turn off SMTP authentication.

smtp_userid

The user ID for SMTP authentication. By default, this parameter takes the same value as the pop3_userid parameter. The smtp_userid only needs to be set if the user ID is different from that of the POP server.

smtp_password

The password for SMTP authentication. By default, this parameter takes the same value as the pop3_password parameter. The smtp_password only needs to be set if the user ID is different from that of the POP server.

smtp_port

The number of the port on which the SMTP server is currently listening. The default is 25. This corresponds to the port field in the SMTP/POP3 login window.

suppress_dialogs

This parameter is set to ON or OFF. The default value is OFF. If it is set to ON, the *Connect* window does not appear after failed attempts to connect to the messaging system. Instead, an error is generated.

Remarks

The SQL Remote (dbremote) Message Agent saves message system parameters when the user enters them in the message system window when the message system is first used. In this case, it is not necessary to use this statement explicitly. This statement is most useful when preparing a consolidated database for extracting many databases.

The option names are case sensitive. The case sensitivity of option values depends on the option: Boolean values are case insensitive, while the case sensitivity of passwords, directory names, and other strings depend on the case sensitivity of the file system (for directory names), or the database (for user IDs and passwords).

Privileges

Publishers can set their own options. Otherwise, you must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement sets the FTP host to `ftp.mycompany.com` for the FTP link for user Sam_Singer:

```
SET REMOTE FTP OPTION Sam_Singer.host = 'ftp.mycompany.com';
```

The following statement stops SQL Remote from using the specified file extensions for messages that are generated:

```
SET REMOTE FTP OPTION "PUBLIC"."invalid_extensions"='exe,pif,dll,bat,cmd,vbs';
```

The following statement sets the URL to point to the localhost for the HTTP link for user Sam_Singer:

```
SET REMOTE HTTP OPTION Sam_Singer.url='localhost:8033';
```

The following statement sets the HTTP URL to point to a Relay Server that forwards the request to the srhttp farm:

```
SET REMOTE HTTP OPTION PUBLIC.url='iis7.company.com:80/rs/client/rs.dll/srhttp';
```

Related Information

Custom Encoding Schemes
The FTP Message System
The FILE Message System
The HTTP Message System
The SMTP Message System
Collecting Errors from the Remote Database (SQL)
Tutorial: Setting up a Replication System Using the HTTP Message System with a Separate Message Server
Remote Message Type Control Parameters
SET OPTION Statement [page 1387]
CREATE CERTIFICATE Statement [page 820]

1.4.4.254 SET SQLCA Statement [ESQL]

Instructs the SQL preprocessor to use a SQLCA other than the default, global `sqlca`.

Syntax

```
SET SQLCA sqlca
```

```
sqlca : identifier | string
```

Remarks

The SET SQLCA statement tells the SQL preprocessor to use a SQLCA other than the default global `sqlca`. The sqlca must be an identifier or string that is a C language reference to a SQLCA pointer.

The current SQLCA pointer is implicitly passed to the database interface library on every Embedded SQL statement. All Embedded SQL statements that follow this statement in the C source file will use the new SQLCA.

This statement is necessary only when you are writing code that is reentrant.

The sqlca should reference a local variable. Any global or module static variable is subject to being modified by another thread.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Each application that calls the ExecuteSQL function in the example below has its own SQLCA which it passes as a parameter. The SET SQLCA statement guarantees that all references to the SQLCA (explicit or implicit) in the statements that follow will refer to the sqlcaptr field of the first parameter.

```
typedef struct an_application_struct
{
    SQLCA   *sqlcaptr;
} an_application;
an_sql_code ExecuteSQL( an_application *app, char *cmd )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char *sqlcommand;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SET SQLCA "app->sqlcaptr";
    sqlcommand = cmd;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL EXECUTE IMMEDIATE :sqlcommand;
    return( SQLCODE );
}
```

Related Information

SQLCA Management for Multithreaded or Reentrant Code

1.4.4.255 SETUSER Statement

Allows a user to assume the identity of (impersonate) another authorized user.

Syntax

```
{ SETUSER | SET SESSION AUTHORIZATION }
[ [ WITH OPTION ] userid ]
```

Parameters

SETUSER or SET SESSION AUTHORIZATION

SETUSER and SET SESSION AUTHORIZATION are semantically equivalent. However, the value you specify for SETUSER must be formatted as an identifier (for example, `SETUSER JoeS` or `SETUSER "JoeS"`), whereas the value you specify for SET SESSION AUTHORIZATION must be a formatted as a string (for example, `SET SESSION AUTHORIZATION 'JoeS'`).

WITH OPTION clause

During an impersonation session, database option settings in effect for the impersonator may be set differently than those of `userid`, which can impact results. Specify WITH OPTION to change the database options to reflect the options in effect for `userid`.

Remarks

The SETUSER statement is provided for administrative use and should not be used for connection pooling. After executing a SETUSER statement, you can execute one of the following commands to verify which user authorization you have assumed:

- SELECT USER
- SELECT CURRENT USER

SETUSER with no user ID undoes all earlier SETUSER statements.

A successful impersonation remains in effect until it is manually terminated (by executing a SETUSER statement with no ID) or the session is terminated.

The SETUSER statement cannot be used inside a procedure, trigger, event handler or batch.

There are several uses for the SETUSER statement, including the following:

Creating objects

You can use SETUSER to create a database object that is to be owned by another user.

Privilege checking

By acting as another user, with their privileges and inheritances, a user can test the privileges and name resolution of queries, procedures, views, and so on.

Providing a safer environment for administrators

The database administrator has permission to perform any action in the database. To ensure that you do not accidentally perform an unintended action, use SETUSER to switch to a different user ID with fewer privileges.

Privileges

You must have the SET USER system privilege. However, your ability to successfully execute a SETUSER statement (start an impersonation session) depends on whether you meet the at-least criteria for the person you are attempting to impersonate. The SETUSER statement fails if this condition is not met.

Standards

ANSI/ISO SQL Standard

The SET SESSION AUTHORIZATION syntax is part of optional ANSI/ISO SQL Language Feature F321, "User authorization". The SETUSER syntax is not part of the Standard. You can use the WITH OPTION syntax with both variants, but WITH OPTION is not part of the Standard.

Example

In the first statement in this example (`SETUSER "Joe"`), a user who has the SET USER system privilege impersonates Joe to run some operations using Joe's privileges. In the second statement (`SETUSER WITH OPTION "Jane"`), the user impersonates Jane to perform some operations using Jane's privileges and the database options currently in effect for Jane. In the third statement (`SETUSER`), the user reverts back to their own user ID, privileges, and database options.

```
SETUSER "Joe"
//  Some operations are run using Joes privileges ...
SETUSER WITH OPTION "Jane"
//  Some operations are run using Jane's privileges, and the
//  database options in effect are changed to the current
//  database options for Jane
SETUSER
```

Related Information

Impersonation
In-depth Look at the Impersonation At-least Criteria
Strings [page 11]
EXECUTE IMMEDIATE Statement [SP] [page 1155]
GRANT Statement [page 1193]
REVOKE Statement [page 1345]
SET OPTION Statement [page 1387]
Identifiers [page 6]

1.4.4.256 SIGNAL Statement [SP]

Signals an exception condition.

Syntax

```
SIGNAL exception-name
```

Remarks

SIGNAL allows you to raise an exception.

Use `exception-name` to specify the name of an exception declared using a DECLARE statement at the beginning of the current compound statement. The exception must correspond to a system-defined SQLSTATE or a user-defined SQLSTATE. User-defined SQLSTATE values must be in the range 99000 to 99999.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The SIGNAL statement is part of optional ANSI/ISO SQL Language Feature P002, "Computational completeness".

Transact-SQL

The SIGNAL statement cannot be used in Transact-SQL compound statements and procedures.

Example

The following compound statement declares and signals a user-defined exception. If you execute this example from Interactive SQL, the message 'My exception signaled' appears on the *History* tab in the *Results* area.

```
BEGIN
   DECLARE myexception EXCEPTION
   FOR SQLSTATE '99001';
   SIGNAL myexception;
   EXCEPTION
      WHEN myexception THEN
         MESSAGE 'My exception signaled'
         TO CLIENT;
END
```

Related Information

Exception Handlers
RESIGNAL Statement [SP] [page 1338]
BEGIN Statement [page 784]
RAISERROR Statement [page 1315]

1.4.4.257 START DATABASE Statement

Starts a database on the current database server.

Syntax

```
START DATABASE database-file [ start-options ... ]
```

```
start-options :
[ AS database-name ]
[ WITH TRUNCATE AT CHECKPOINT ]
[ FOR READ ONLY ]
[ AUTOSTOP { ON | OFF } ]
[ KEY key ]
[ WITH SERVER NAME alternative-database-server-name ]
[ DIRECTORY dbspace-directory ]
[ CHECKSUM { ON | OFF } ]
[ DISKSANDBOX { ON | OFF | DEFAULT } ]
[ MIRROR ON ]
```

Parameters

database-file

The `database-file` parameter is a string. If a relative path is supplied in `database-file`, it is relative to the database server starting directory.

start-options clause

The `start-options` clauses can be listed in any order:

AS clause

If `database-name` is not specified, a default name is assigned to the database. This default name is the root of the database file. For example, a database in file `C:\Database Files\`*demo.db* would be given the default name of demo. The `database-name` parameter is an identifier.

WITH TRUNCATE AT CHECKPOINT clause

Starts a database with log truncation on checkpoint enabled.

FOR READ ONLY clause

Starts a database in read-only mode. When used on a database requiring recovery, the statement fails and the error message is returned.

AUTOSTOP clause

The default setting for the AUTOSTOP clause is ON. With AUTOSTOP set to ON, the database is unloaded when the last connection to it is dropped. If AUTOSTOP is set to OFF, the database is not unloaded.

In Interactive SQL, you can use YES or NO as alternatives to ON and OFF.

KEY clause

If the database is strongly encrypted, enter the KEY value (password) using this clause.

WITH SERVER NAME clause

Use this clause to specify an alternate name for the database server when connecting to this database.

Do not use this clause with mirrored databases.

DIRECTORY clause

Use this clause to specify the directory where the dbspace files are located for the database that is being started. For example, if the database server is started in the same directory as the dbspaces, and you include the `DIRECTORY '.'` clause, then this instructs the database server to find all dbspaces in the current directory.

CHECKSUM clause

Use this clause to enable write checksums for newly written pages for databases that were not created with global checksums enabled. This clause has the same behavior as the -wc database option.

The difference between the CHECKSUM clause and creating a database with global checksums enabled is that when you specify CHECKSUM ON, database pages are checksummed only when they are written out to disk. Pages that are read from disk are only verified if a checksum value was calculated before the pages were written. If a database has global checksums enabled, checksums are calculated for all pages when they are written and checksums are verified for all pages when they are read.

If the database server detects that the database is running on a removable storage device, such as a network share or USB device, then the database server automatically enables write checksums for all database pages.

By default, databases created with version 10 and 11 of SQL Anywhere do not have global checksums enabled. If you start a database created with SQL Anywhere 10 or 11 on a version 12 or later database server, then by default the database server creates write checksums for pages when they are written to disk (CHECKSUM ON). Version 12 and later databases have global checksums enabled by default, so the database server defaults to CHECKSUM OFF for these databases because by default all database pages have checksums. You can use either the -wc option or the START DATABASE statement to change the database server's checksum behavior if you do not want to use the default checksum settings.

You can check whether a database was created with global checksums enabled by executing the following statement:

```
SELECT DB_PROPERTY ( 'Checksum' );
```

You can check whether write checksums are enabled by executing the following statement:

```
SELECT DB_PROPERTY ( 'WriteChecksum' );
```

DISKSANDBOX clause

Set DISKSANDBOX to ON to restrict read-write file operations on the database to the directory where the main database file is located. Set DISKSANDBOX to OFF to allow access to all directories. If DISKSANDBOX is set to DEFAULT, the disk sandbox settings specified by the -sbx database server option are used.

> **i Note**
>
> If you start a database server with the -sbx database server option, then you must provide the secure feature key for the manage_disk_sandbox secure feature to start a database with DISKSANDBOX OFF.

MIRROR ON clause

Use the MIRROR ON clause to add an additional mirrored database to database servers that are already running and possibly hosting a mirrored database. You must specify the AUTOSTOP OFF clause when using this clause.

Remarks

Starts a specified database on the current database server.

The START DATABASE statement does not connect the current application to the specified database: an explicit connection is still needed.

If you are not connected to a database and you want to use the START DATABASE statement, you must first connect to a database, such as the utility database.

You can only use the database name utility_db to connect to the SQL Anywhere utility database.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

Privileges

The required privileges to start a database on a network server are specified by the database server -gd option. By default, the SERVER OPERATOR system privilege is required to start a database on the network server.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example starts a fictitious database file `C:\temp\sample_2.db` on the current server:

```
START DATABASE 'c:\\temp\\sample_2.db';
```

This example starts same database but as sam2:

```
START DATABASE 'c:\\temp\\sample_2.db'
AS sam2;
```

Related Information

The Utility Database (utility_db)
Corruption Detection Using Checksums
Adding a Mirrored Database to a Running Mirroring System
-sbx Database Server Option
STOP DATABASE Statement [page 1421]
CREATE MIRROR SERVER Statement [page 901]
CONNECT Statement [ESQL] [Interactive SQL] [page 815]
VALIDATE Statement [page 1476]
-gd Database Server Option
-ds Database Option
-ek Database Option
-m Database Option
-r Database Option
-n Database Option
-wc Database Option
-sn Database Option

1.4.4.258 START EXTERNAL ENVIRONMENT Statement

Starts an external environment.

Syntax

```
START EXTERNAL ENVIRONMENT environment-name
```

```
environment-name :
C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64
| CLR
| JAVA
| JS
```

| *PERL*
| *PHP*

Parameters

environment-name

The name of the external environment to start.

Remarks

The START EXTERNAL ENVIRONMENT statement can be used to ensure that the external environment module can be located and started. Since an external environment is automatically started, this statement is not required.

If the CLR external environment cannot be started, make sure the corresponding Sap.Data.SQLAnywhere assembly is installed using SetupVSPackage. For example, `Sap.Data.SQLAnywhere.v4.5.dll` must be installed when the following statement has been executed.

```
ALTER EXTERNAL ENVIRONMENT CLR LOCATION 'dbextclr[VER_MAJOR]_v4.5';
```

Privileges

None

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Start the Perl external environment.

```
START EXTERNAL ENVIRONMENT PERL;
```

Related Information

External Environment Support
ALTER EXTERNAL ENVIRONMENT Statement [page 680]
STOP EXTERNAL ENVIRONMENT Statement [page 1423]
INSTALL EXTERNAL OBJECT Statement [page 1238]
REMOVE EXTERNAL OBJECT Statement [page 1333]
SYSEXTERNENV System View [page 1908]
SYSEXTERNENVOBJECT System View [page 1910]

1.4.4.259 START JAVA Statement

Starts the Java VM.

Syntax

START JAVA

Remarks

The START JAVA statement starts the Java VM. The main use is to load the Java VM at a convenient time so that when the user starts to use Java functionality there is no initial pause while the Java VM is loaded.

The database server must be set up to locate a Java VM. Since you can specify different Java VMs for each database, the ALTER EXTERNAL ENVIRONMENT statement can be used to indicate the location (path) of the Java VM.

A Java VM must be installed.

Privileges

None

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example starts the Java VM.

```
START JAVA;
```

Related Information

How to Start and Stop the Java VM
STOP JAVA Statement [page 1424]
java_location Option

1.4.4.260 START LOGGING Statement [Interactive SQL]

Starts logging executed SQL statements and messages to a log file.

Syntax

```
START LOGGING filename
```

Remarks

The START LOGGING statement starts copying all subsequent executed SQL statements and messages to the log file that you specify. If the file does not exist, Interactive SQL creates it. If the file does exist, Interactive SQL will append to it. Logging continues until you explicitly stop the logging process with the STOP LOGGING statement, or until you end the current Interactive SQL session.

You can also start and stop logging by clicking SQL > Start Logging and SQL > Stop Logging.

Execution times are included in the log file when logging and execution time reporting are both enabled.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example starts logging to a file `c:\temp\sql.log`:

```
START LOGGING 'c:\\temp\\sql.log';
```

Related Information

Logging Statements (Interactive SQL)
STOP LOGGING Statement [Interactive SQL] [page 1426]
echo Option [Interactive SQL]
isql_command_timing Option [Interactive SQL]

1.4.4.261 START SERVER Statement [Interactive SQL]

Starts a database server.

Syntax

```
START SERVER AS database-server-name [ STARTLINE command-string ]
```

Remarks

The START SERVER statement starts a database server. To specify a set of options for the database server, use the STARTLINE keyword together with a command string.

START ENGINE is accepted for compatibility reasons, but is deprecated.

This SQL statement is not supported for SAP HANA databases.

Privileges

None

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example starts a database server named sample without starting any databases on it.

```
START SERVER AS sample;
```

This example shows the use of a STARTLINE clause.

```
START SERVER AS eng1 STARTLINE 'dbsrv17 -c 8M';
```

Related Information

Interactive SQL
STOP SERVER Statement [page 1427]
SQL Anywhere Database Server Executable (dbsrv17, dbeng17)

1.4.4.262 START SUBSCRIPTION Statement [SQL Remote]

Starts a subscription for a user to a publication.

Syntax

```
START SUBSCRIPTION
TO publication-name [ ( subscription-value ) ]
FOR subscriber-id, ...
```

Parameters

publication-name

The name of the publication to which the user is being subscribed. This may include the owner of the publication.

subscription-value

A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.

subscriber-id

The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Remarks

A SQL Remote subscription starts when publication updates are being sent from the consolidated database to the remote database.

The START SUBSCRIPTION statement is one of a set of statements that manage subscriptions. The CREATE SUBSCRIPTION statement defines the data that the subscriber is to receive. The SYNCHRONIZE

SUBSCRIPTION statement ensures that the consolidated and remote databases are consistent with each other. The START SUBSCRIPTION statement is required to start messages being sent to the subscriber.

Data at each end of the subscription must be consistent before a subscription is started. Use the database extraction utility to manage the creation, synchronization, and starting of subscriptions. If you use the database extraction utility, you do not need to execute an explicit START SUBSCRIPTION statement. Also, the Message Agent starts subscriptions once they are synchronized.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement starts the subscription of user Sam_Singer to the pub_contact publication.

```
START SUBSCRIPTION TO pub_contact
FOR Sam_Singer;
```

Related Information

Starting Subscriptions (SQL Central)
Creating Multiple Remote Databases (Command Line)
CREATE SUBSCRIPTION Statement [SQL Remote] [page 993]
REMOTE RESET Statement [SQL Remote] [page 1332]
SYNCHRONIZE SUBSCRIPTION Statement [SQL Remote] [page 1438]
STOP SUBSCRIPTION Statement [SQL Remote] [page 1428]
Extraction Utility (dbxtract)

1.4.4.263 START SYNCHRONIZATION DELETE Statement [MobiLink]

Restarts logging of deletes for MobiLink synchronization.

Syntax

START SYNCHRONIZATION DELETE

Remarks

Ordinarily, SQL Anywhere and UltraLite automatically log any changes made to tables or columns that are part of a synchronization, and upload these changes to the consolidated database during the next synchronization. You can temporarily suspend automatic logging of delete operations using the STOP SYNCHRONIZATION DELETE statement. The START SYNCHRONIZATION DELETE statement allows you to restart the automatic logging.

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the delete operations executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed. Repeating STOP SYNCHRONIZATION DELETE has no additional effect.

A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

Do not use START SYNCHRONIZATION DELETE if your application does not synchronize data.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE:

```
-- Prevent deletes from being sent
-- to the consolidated database
STOP SYNCHRONIZATION DELETE;
-- Remove all records older than 1 month
-- from the remote database,
-- NOT the consolidated database
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT TIMESTAMP, -1 )
-- Re-enable all deletes to be sent
-- to the consolidated database
-- DO NOT FORGET to start this
START SYNCHRONIZATION DELETE;
-- Commit the entire operation,
-- otherwise rollback everything
-- including the stopping of the deletes
commit;
```

Related Information

STOP SYNCHRONIZATION DELETE Statement [MobiLink] [page 1430]

1.4.4.264 START SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink]

Starts a MobiLink synchronization schema change.

Syntax

```
START SYNCHRONIZATION SCHEMA CHANGE
FOR TABLES table-list
set-script-version
  | set-script-version-on-subscription, ...
```

```
set-script-version :
SET SCRIPT VERSION = script-version
```

```
set-script-version-on-subscription :
SET SCRIPT VERSION = script-version ON SUBSCRIPTION subscription_name
```

```
script-version : string | NULL
```

```
subscription-name : identifier
```

Parameters

FOR TABLES clause

This clause specifies the tables that are affected by the schema change.

SET SCRIPT VERSION clause

Specifies the new script version for all subscriptions that contain any table specified in the FOR TABLES clause. The new script version may be the same as the existing script version.

SET SCRIPT VERSION...ON SUBSCRIPTION clause

Specifies the new script version for the specified subscription. When used, this clause must be repeated, separated by commas, for each subscription that contains any table specified in the FOR TABLES clause. The new script version may be the same as the existing script version.

Remarks

All tables to which you want to apply a schema change must be listed in `table-list`. A table cannot be listed more than once. An error message is reported if there is an existing lock on any of the tables in `table-list`.

Only one synchronization schema change can be executed on a database at a time. The START SYNCHRONIZATION SCHEMA CHANGE statement fails when another schema change is in progress.

The database server obtains locks on all tables specified in `table-list`. The database server ignores the setting of the blocking option when attempting to obtain locks. If a lock cannot be obtained, all previously acquired locks are released and an error message is reported.

During a synchronization schema change:

- You cannot execute a data manipulation statement.
- You cannot execute additional START SYNCHRONIZATION SCHEMA CHANGE statements.
- You can alter a publication to change the column subsetting of any table in `table-list`.
- You can alter a publication to drop any table in `table-list`.
- You can alter any of the tables listed in `table-list`.

An implicit commit is performed both before and after the START SYNCHRONIZATION SCHEMA CHANGE statement is executed. A synchronization schema change ends with the execution of a STOP SYNCHRONIZATION SCHEMA CHANGE statement. When the STOP SYNCHRONIZATION SCHEMA CHANGE statement is executed, all table locks are released.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION SCHEMA CHANGE and STOP SYNCHRONIZATION SCHEMA CHANGE:

```
START SYNCHRONIZATION SCHEMA CHANGE
  FOR TABLES GROUPO.SalesOrders, GROUPO.Products
  SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub1,
  SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub2;
ALTER TABLE GROUPO.SalesOrders ADD SUBTOTAL NUMERIC (10,2);
ALTER TABLE GROUPO.Products ALTER QUANTITY BIGINT;
STOP SYNCHRONIZATION SCHEMA CHANGE;
```

Related Information

STOP SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink] [page 1432]

1.4.4.265 STOP DATABASE Statement

Stops a database on the current database server.

Syntax

```
STOP DATABASE database-name
[ ON database-server-name ]
[ UNCONDITIONALLY ]
```

Parameters

STOP DATABASE clause

The `database-name` is the name of a database (other than the current database) running on the current server.

ON clause

This clause is supported in Interactive SQL only. If `database-server-name` is not specified in Interactive SQL, all running servers are searched for a database of the specified name.

When not using this statement in Interactive SQL, the database is stopped only if it is started on the current database server.

UNCONDITIONALLY clause

Stop the database even if there are connections to the database. By default, the database is not stopped if there are connections to it.

Remarks

The STOP DATABASE statement stops a specified database on the current database server.

You cannot use STOP DATABASE on the database to which you are currently connected.

If you want the database server to ignore certain connections when determining whether or not to shut down the database, then you can use the connection_type option.

Privileges

The required privileges to stop a database on the network server are determined by the database server -gd option. The default system privilege for stopping a database on the network server is SERVER OPERATOR.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Stop the database named `sample` on the current server.

```
STOP DATABASE sample;
```

Related Information

START DATABASE Statement [page 1407]
DISCONNECT Statement [ESQL] [Interactive SQL] [page 1085]
-gd Database Server Option
Stop Server Utility (dbstop)

1.4.4.266 STOP EXTERNAL ENVIRONMENT Statement

Stops an external environment.

Syntax

```
STOP EXTERNAL ENVIRONMENT environment-name
```

```
environment-name :
C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64
| CLR
| JAVA
| JS
| PERL
| PHP
```

Parameters

environment-name

The name of the external environment to stop.

Remarks

None.

Privileges

None

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example stops the Perl external environment.

```
STOP EXTERNAL ENVIRONMENT PERL;
```

Related Information

External Environment Support
ALTER EXTERNAL ENVIRONMENT Statement [page 680]
START EXTERNAL ENVIRONMENT Statement [page 1410]
INSTALL EXTERNAL OBJECT Statement [page 1238]
REMOVE EXTERNAL OBJECT Statement [page 1333]
SYSEXTERNENV System View [page 1908]
SYSEXTERNENVOBJECT System View [page 1910]

1.4.4.267 STOP JAVA Statement

Stops the Java VM.

Syntax

STOP JAVA

Remarks

The STOP JAVA statement unloads the ClassLoader for the current connection. If the current connection is the last connection using the Java VM, then the STOP JAVA statement will also exit the Java VM. The main use of the STOP JAVA statement is to economize on the use of system resources.

Privileges

None.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

This example stops the Java VM.

```
STOP JAVA;
```

Related Information

START JAVA Statement [page 1412]

1.4.4.268 STOP LOGGING Statement [Interactive SQL]

Stops logging of SQL statements and messages for the current session.

Syntax

STOP LOGGING

Remarks

The STOP LOGGING statement stops Interactive SQL from logging SQL statements and messages to the log file. You can start logging with the START LOGGING statement.

You can also stop logging by clicking *SQL* > *Stop Logging*.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example stops the current logging session.

```
STOP LOGGING;
```

Related Information

Logging Statements (Interactive SQL)
START LOGGING Statement [Interactive SQL] [page 1413]
echo Option [Interactive SQL]

1.4.4.269 STOP SERVER Statement

Stops a database server.

Syntax

```
STOP SERVER [ database-server-name ] [ UNCONDITIONALLY ]
```

Parameters

UNCONDITIONALLY clause

If you are the only connection to the database server, you do not need to use UNCONDITIONALLY. If there are other connections, the database server stops only if you use the UNCONDITIONALLY keyword.

Remarks

You can only use `database-server-name` in Interactive SQL only. If you do not execute this statement from Interactive SQL, the current database server is stopped.

By default, the database server is not stopped if there are other connections to it. If the UNCONDITIONALLY clause is used, the database server is stopped even if there are other connections to the database server. If the STOP SERVER statement is executed on a client connection and the server successfully shuts down, a communication error occurs.

If you want the database server to ignore certain connections when determining whether or not to shut down the database, then you can use the connection_type option.

The STOP SERVER statement cannot be used in stored procedures, triggers, events, or batches.

STOP ENGINE is accepted for compatibility reasons, but is deprecated.

Privileges

The privileges to shut down a network server (dbsrv17) depend on the -gk setting on the database server command line. By default, the SERVER OPERATOR privilege is required to shut down a network server.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Stop the current database server, as long as there are no other connections.

```
STOP SERVER;
```

Related Information

START SERVER Statement [Interactive SQL] [page 1415]
-gk Database Server Option

1.4.4.270 STOP SUBSCRIPTION Statement [SQL Remote]

Stops a subscription for a user to a publication.

Syntax

```
STOP SUBSCRIPTION
TO publication-name [ ( subscription-value ) ]
FOR subscriber-id, ...
```

Parameters

publication-name

The name of the publication to which the user is being subscribed. This may include the owner of the publication.

subscription-value

A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.

subscriber-id

The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Remarks

A SQL Remote subscription starts when publication updates are being sent from the consolidated database to the remote database.

The STOP SUBSCRIPTION statement prevents any further messages being sent to the subscriber. The START SUBSCRIPTION statement is required to restart messages being sent to the subscriber. Ensure that the subscription is properly synchronized before restarting: that no messages have been missed.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement stops the subscription of user SamS to the pub_contact publication.

```
STOP SUBSCRIPTION TO pub_contact
FOR Sam_Singer;
```

Related Information

Stopping Subscriptions (SQL Central)
DROP SUBSCRIPTION Statement [SQL Remote] [page 1128]
START SUBSCRIPTION Statement [SQL Remote] [page 1416]
SYNCHRONIZE SUBSCRIPTION Statement [SQL Remote] [page 1438]
Extraction Utility (dbxtract)

1.4.4.271 STOP SYNCHRONIZATION DELETE Statement [MobiLink]

Temporarily stops logging of deletes for MobiLink synchronization.

Syntax

STOP SYNCHRONIZATION DELETE

Remarks

Ordinarily, SQL Anywhere and UltraLite remote databases automatically log any changes made to tables or columns that are being synchronized, and then upload these changes to the consolidated database during the next synchronization. This statement allows you to temporarily suspend logging of delete operations to a SQL Anywhere or UltraLite remote database.

None of the delete operations executed on a connection between the time the connection executes STOP SYNCHRONIZATION DELETE and the time the connection executes START SYNCHRONIZATION DELETE are synchronized.

A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

This statement can be useful to make corrections to a remote database, but should be used with caution as it effectively disables MobiLink synchronization.

Do not use STOP SYNCHRONIZATION DELETE if your application does not synchronize data.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE:

```
-- Prevent deletes from being sent
-- to the consolidated database
STOP SYNCHRONIZATION DELETE;
-- Remove all records older than 1 month
-- from the remote database,
-- NOT the consolidated database
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT TIMESTAMP, -1 )
-- Re-enable all deletes to be sent
-- to the consolidated database
-- DO NOT FORGET to start this
START SYNCHRONIZATION DELETE;
-- Commit the entire operation,
-- otherwise rollback everything
-- including the stopping of the deletes
commit;
```

Related Information

START SYNCHRONIZATION DELETE Statement [MobiLink] [page 1418]

1.4.4.272 STOP SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink]

Stops a MobiLink synchronization schema change.

Syntax

STOP SYNCHRONIZATION SCHEMA CHANGE

Remarks

The STOP SYNCHRONIZATION SCHEMA CHANGE statement stops a schema change started by a START SYNCHRONIZATION SCHEMA CHANGE statement. All locks obtained by the START SYNCHRONIZATION SCHEMA CHANGE statement are released.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION SCHEMA CHANGE and STOP SYNCHRONIZATION SCHEMA CHANGE:

```
START SYNCHRONIZATION SCHEMA CHANGE
  FOR TABLES GROUPO.SalesOrders, GROUPO.Products
  SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub1,
  SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub2;
```

```
ALTER TABLE GROUPO.SalesOrders ADD SUBTOTAL NUMERIC (10,2);
ALTER TABLE GROUPO.Products ALTER QUANTITY BIGINT;
STOP SYNCHRONIZATION SCHEMA CHANGE;
```

Related Information

START SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink] [page 1419]
SYSARTICLE System View [page 1898]

1.4.4.273 SYNCHRONIZE Statement [MobiLink]

Synchronizes a SQL Anywhere database with a MobiLink server. The synchronization options can be specified in the statement itself.

Syntax

```
SYNCHRONIZE {
PROFILE sync-profile-name [ MERGE sync-option [ ;... ] ]
| USING sync-option [ ;... ]
| START
| STOP
}
```

```
[ PORT port-number ]
[ VERBOSITY { LOW | NORMAL | HIGH } ]
[ TIMEOUT timeout ]
[ USER user-name IDENTIFIED BY password ]
[ KEY key]
```

```
sync-option : string
```

Parameters

sync-profile-name

The name of the synchronization profile to use for this synchronization.

MERGE clause

Use this clause to add or override synchronization profile options.

USING clause

Use this clause to specify synchronization profile options when you are not using a synchronization profile.

sync-option

A string of one or more synchronization profile option value pairs, separated by semicolons. For example, `'option1=value1;option2=value2'`.

START clause

Starts the dbmlsync utility running in server mode and leaves it running. No synchronization is performed. When you are performing more than one synchronization in a short period, you can improve performance by explicitly starting the dbmlsync server using this clause, performing your synchronizations, then explicitly stopping the dbmlsync server using the STOP clause.

If the database is strongly encrypted, and you are using the SYNCHRONIZE START clause to prestart the dbmlsync utility in server mode, then subsequent calls to SYNCHRONIZE PROFILE or SYNCHRONIZE STOP do not need to specify the KEY clause.

STOP clause

Stops a dbmlsync server that was previously started using the START clause. No synchronization is performed.

PORT clause

Use this clause to specify the port number that the database server uses to communicate with the dbmlsync utility. The default is 4433.

VERBOSITY clause

This clause controls the amount of information that is added to the synchronize_results and synchronize_parameters shared global temporary tables during synchronization.

The following is a list of client API events that are returned by each VERBOSITY option.

Option	Returns
LOW	• DBSC_EVENTTYPE_SYNC_START • DBSC_EVENTTYPE_SYNC_DONE • DBSC_EVENTTYPE_ERROR_MSG • DBSC_EVENTTYPE_WARNING_MSG
NORMAL (default)	• DBSC_EVENTTYPE_SYNC_START • DBSC_EVENTTYPE_SYNC_DONE • DBSC_EVENTTYPE_ERROR_MSG • DBSC_EVENTTYPE_WARNING_MSG • DBSC_EVENTTYPE_INFO_MSG
HIGH	• DBSC_EVENTTYPE_SYNC_START • DBSC_EVENTTYPE_SYNC_DONE • DBSC_EVENTTYPE_ERROR_MSG • DBSC_EVENTTYPE_WARNING_MSG • DBSC_EVENTTYPE_INFO_MSG • DBSC_EVENTTYPE_PROGRESS_INDEX • DBSC_EVENTTYPE_PROGRESS_TEXT • DBSC_EVENTTYPE_TITLE

Be careful not to confuse the VERBOSITY clause of the SYNCHRONIZE statement with the VERBOSITY option that you can specify in a synchronization profile. The VERBOSITY clause of the SYNCHRONIZE statement controls the type of events that are recorded in the synchronize_results and

synchronize_parameters tables. The VERBOSITY option in a synchronization profile controls the number of DBSC_EVENTTYPE_INFO_MSG events that are generated during synchronization.

```
SYNCHRONIZE PROFILE SalesData VERBOSITY NORMAL;
```

```
SYNCHRONIZE PROFILE SalesData MERGE 'Verbosity=BASIC,ROW_DATA' VERBOSITY
NORMAL;
```

TIMEOUT clause

This clause specifies how long the database server waits, in seconds, for the synchronization to complete before attempting to cancel the synchronization. The default is 240 seconds.

USER/IDENTIFIED BY clause

Use this clause to specify the database user ID and password that the dbmlsync utility uses to connect to the remote database to perform the synchronization. The user ID specified must have the SYS_RUN_REPLICATION_ROLE system role. By default, synchronization uses the user ID for the database connection that executed the SYNCHRONIZE statement.

KEY clause

If the database is strongly encrypted, then you must specify the database encryption key in the SYNCHRONIZE command so that the transaction logs can be unencrypted. If the database is strongly encrypted and no KEY clause is specified, then you are prompted for the database encryption key.

Remarks

This statement can only be used if the MobiLink client for SQL Anywhere including the Dbmlsync C++ API is installed.

The MobiLink client for SQL Anywhere is not available on all platforms where the database server may run.

When synchronization is complete, you can view the results of the synchronization in the synchronize_results and synchronize_parameters shared global temporary tables. The synchronize_results and synchronize_parameters tables store the results of all synchronizations that have been executed with the SYNCHRONIZE statement since the database server was started. The synchronize_results and synchronize_parameters tables are truncated each time the database server is shut down.

The synchronize_results table contains the following columns:

Column name	Data type	Description
row_id	UNSIGNED BIGINT	The primary key of the table used to determine the order in which rows were inserted into the table.
conn_id	UNSIGNED INT	The connection id number of the connection that executed the SYNCHRONIZE statement that generated this event.
result_time	TIMESTAMP	The time the event was added to the synchronize_results table.

Column name	Data type	Description
result_type	CHAR(128)	The type of event.

Each event shown in the synchronize_results table has 0 or more parameters associated with it that contain additional information about the event. The parameters are stored in the synchronize_parameters table, which contains the following columns:

Column name	Data type	Description
row_id	UNSIGNED BIGINT	A foreign key to the row_id column in the synchronize_results table. Use this value to match each parameter back to the event to which it belongs.
parm_id	UNSIGNED INT	Contains the numeric ID of the parameter. For events with more than 1 parameter, use this value to locate the specific parameter you need.
parm_message	LONG VARCHAR	The value associated with the parameter.

To view information about past or current synchronizations, you can use the sp_get_last_synchronize_result system procedure as an alternative to directly querying the synchronize_results and synchronize_parameters tables.

Alternately, you can use the following statement to view the results of all the synchronizations that have taken place since the database server was started.

```
SELECT *
     FROM synchronize_results sr
     KEY JOIN synchronize_parameters sp
     ORDER BY sr.row_id , sp.parm_id
```

You can use the synchronize_results and synchronize_parameters tables to monitor the progress of a synchronization on a connection that is different from your current connection. To monitor the progress of a synchronization on a different connection:

- Execute a SELECT CONNECTION_PROPERTY statement to determine the connection ID of the current connection.
- Execute a SYNCHRONIZE statement to start synchronization.
- On a separate connection, use the sp_get_last_synchronize_results system procedure to retrieve results using the connection ID you determined above.

To view the results of a synchronization that is complete or in progress on a specific connection, you can use the sp_get_last_synchronize_results system procedure.

The SYNCHRONIZE statement is similar to the UltraLite SYNCHRONIZE statement. However, the SQL Anywhere SYNCHRONIZE statement launches the dbmlsync utility in server mode to perform the synchronization. The UltraLite SYNCHRONIZE statement uses UltraLite runtime.

In cases where there are multiple versions of SQL Anywhere installed on the same system, or if the dbmlsync executable is not located in the same directory as the database server and cannot be found in the PATH environment variable, use the ALTER EXTERNAL ENVIRONMENT command to specify the location of the dbmlsync executable.

The database server functions as a dbmlsync API client and uses TCP/IP to communicate with a dbmlsync server. By default, this communication occurs on port 4433. Use the PORT clause to specify a different port.

Use the SYNCHRONIZE PROFILE and SYNCHRONIZE USING statements to perform a synchronization. Use the SYNCHRONIZE START and SYNCHRONIZE STOP statements to start or stop a dbmlsync server. When executing a SYNCHRONIZE PROFILE or SYNCHRONIZE USING statement, the database server attempts to connect to a dbmlsync server that is already running. If a dbmlsync server that is already running cannot be located, a dbmlsync server is started. When the synchronization is complete, the database server shuts down the dbmlsync server it started. If the statement connected to a dbmlsync server that was already running, the dbmlsync server is not shut down. If you are performing multiple synchronizations and do not want to start and stop the dbmlsync server for each synchronization, you can execute a SYNCHRONIZE START statement, followed by multiple SYNCHRONIZE PROFILE or SYNCHRONIZE USING statements, and end with a SYNCHRONIZE STOP statement.

If you use this statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have either the MANAGE REPLICATION system privilege or the SYS_RUN_REPLICATION_ROLE system role.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example shows the syntax for synchronizing a synchronization profile named Test1:

```
SYNCHRONIZE PROFILE Test1;
```

Related Information

MobiLink Synchronization
MobiLink Synchronization Profiles
CREATE SYNCHRONIZATION PROFILE Statement [MobiLink] [page 996]
SQL Anywhere MobiLink Client Deployment
sp_get_last_synchronize_result System Procedure [page 1779]
ALTER EXTERNAL ENVIRONMENT Statement [page 680]

1.4.4.274 SYNCHRONIZE SUBSCRIPTION Statement [SQL Remote]

Synchronizes a subscription for a user to a publication.

Syntax

```
SYNCHRONIZE SUBSCRIPTION
TO publication-name [ ( subscription-value ) ]
FOR remote-user, ...
```

Parameters

publication-name

The name of the publication to which the user is being subscribed. This may include the owner of the publication.

subscription-value

A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.

remote-user

The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Remarks

A SQL Remote subscription is **synchronized** when the data in the remote database is consistent with that in the consolidated database, so that publication updates sent from the consolidated database to the remote database will not result in conflicts and errors.

To synchronize a subscription, a copy of the data in the publication at the consolidated database is sent to the remote database. The SYNCHRONIZE SUBSCRIPTION statement does this through the message system.

Where possible, you use the database extraction utility (dbxtract) instead to synchronize subscriptions without using a message system.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement synchronizes the subscription of user Sam_Singer to the pub_contact publication.

```
SYNCHRONIZE SUBSCRIPTION
   TO pub_contact
   FOR Sam_Singer;
```

Related Information

Subscription Resynchronization
Synchronizing with the SQL Remote Message Agent (dbremote)
Synchronizing Subscriptions (SQL Central)
CREATE SUBSCRIPTION Statement [SQL Remote] [page 993]
START SUBSCRIPTION Statement [SQL Remote] [page 1416]
STOP SUBSCRIPTION Statement [SQL Remote] [page 1428]
Extraction Utility (dbxtract)

1.4.4.275 SYSTEM Statement [Interactive SQL]

Launches an executable file from within Interactive SQL.

Syntax

```
SYSTEM '[ path ] filename'
```

Remarks

Launches the specified executable file. The path and file name must be enclosed in single quotation marks.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement launches the Notepad program if the Notepad executable is in your path.

```
SYSTEM 'notepad.exe';
```

Related Information

Interactive SQL

CONNECT Statement [ESQL] [Interactive SQL] [page 815]
Interactive SQL Utility (dbisql)

1.4.4.276 TRIGGER EVENT Statement

Triggers a named event. The event may be defined for event triggers or be a scheduled event.

Syntax

```
TRIGGER EVENT event-name [ ( parm = value, ... ) ]
```

Parameters

parm = value

When a triggering condition causes an event handler to execute, the database server can provide context information to the event handler using the event_parameter function. The TRIGGER EVENT statement allows you to explicitly supply these parameters, to simulate a context for the event handler.

Remarks

Actions are tied to particular trigger conditions or schedules by a CREATE EVENT statement. You can use the TRIGGER EVENT statement to force the event handler to execute, even when the scheduled time or trigger condition has not occurred. TRIGGER EVENT does not execute disabled event handlers.

Each `value` is a string. The maximum length of each `value` is limited by the maximum page size specified by the -gp server option. If the length of `value` exceeds the page size, the string is truncated at the point at which the page is full.

Privileges

You must be the owner of the event, or have the MANAGE ANY EVENT system privilege.

Side Effects

None

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example shows how to pass a string parameter to an event. The event displays the time it was triggered in the database server messages window.

```
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' || event_parameter( 'time' );
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string( current timestamp ) );
```

or, using `keyword=>value` syntax:

```
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' ||
event_parameter( 'what_time' );
END;
TRIGGER EVENT ev_PassedParameter( what_time => string( current timestamp ) );
```

Related Information

-gp Database Server Option
ALTER EVENT Statement [page 675]
CREATE EVENT Statement [page 847]
EVENT_PARAMETER Function [System] [page 372]

1.4.4.277 TRUNCATE Statement

Deletes all rows from a table without deleting the table definition.

Syntax

```
TRUNCATE
TABLE [ owner.]table-name
| MATERIALIZED VIEW [ owner.]materialized-view-name
```

Remarks

The TRUNCATE statement deletes all rows from the table or materialized view.

> **i** Note
>
> The TRUNCATE TABLE statement should be used with great care on a database involved in synchronization or replication because the statement deletes all rows from a table, similar to a DELETE statement that doesn't have a WHERE clause. However, no triggers are fired as a result of a TRUNCATE statement. If the database server determines that a fast truncate is possible, the row deletions are not entered into the transaction log and therefore are not synchronized or replicated. This can lead to inconsistencies that can cause synchronization or replication to fail.

After a TRUNCATE statement, the object's schema and all the indexes continue to exist until you execute a DROP statement. The schema definitions and constraints remain intact, and triggers and privileges remain in effect.

`table-name` can be the name of a base table or a temporary table.

With TRUNCATE TABLE, if all the following criteria are satisfied, the database server performs a fast form of table truncation (fast truncate):

- There are no foreign keys either to or from the table.
- The TRUNCATE TABLE statement is not executed within a trigger.
- The TRUNCATE TABLE statement is not executed within an atomic statement.

If a fast truncation is carried out, individual DELETEs are not recorded in the transaction log, and a COMMIT is carried out before and after the operation. If a fast truncation is not possible, individual DELETES are recorded in the transaction log.

Fast truncation cannot be used within snapshot transactions.

If you attempt to use TRUNCATE TABLE on a table on which an immediate text index is built, or that is referenced by an immediate view, the truncation fails. This does not occur for non-immediate text indexes or materialized views; however, it is strongly recommended that you truncate the data in dependent indexes and materialized views before executing the TRUNCATE TABLE statement on a table, and then refreshing the indexes and materialized views after.

For base tables and materialized views, the TRUNCATE statement requires exclusive access to the table, as the operation is atomic (either all rows are deleted, or none are). Any cursors that were previously opened and that reference the table being truncated must be closed and a COMMIT or ROLLBACK must be executed to release the reference to the table.

For temporary tables, each user has their own copy of the data, and exclusive access is not required when executing the TRUNCATE statement.

Privileges

To execute this statement, one of the following conditions must be true:

- You must be the owner of the table.

- You have ALTER privilege on the table
- You have TRUNCATE privilege on the table
- You have the ALTER ANY TABLE system privilege
- You have the TRUNCATE ANY TABLE system privilege
- You have the ALTER ANY OBJECT system privilege

If the table has foreign keys, then any one of the ALTER privileges is required.

Side Effects

- When you truncate a materialized view, you change the status of the view to uninitialized.
- Delete triggers are not fired by the TRUNCATE statement.
- A COMMIT is performed before and after a TRUNCATE statement is executed.
- Individual deletions of rows are not entered into the transaction log, so the TRUNCATE operation is not replicated. Do not use this statement in SQL Remote replication or on a MobiLink remote database.
- If the table contains a column defined as DEFAULT AUTOINCREMENT or DEFAULT GLOBAL AUTOINCREMENT, the truncation operation resets the next available value for the column.

Standards

ANSI/ISO SQL Standard

The TRUNCATE TABLE statement is optional Language Feature F200. TRUNCATE MATERIALIZED VIEW is not in the standard.

Example

Delete all rows from the SalesOrderItems table:

```
TRUNCATE TABLE GROUPO.SalesOrderItems;
```

Related Information

Deletion of All Rows from a Table
Snapshot Isolation
Advanced: Status and Properties for Materialized Views
DELETE Statement [page 1070]
TRUNCATE TEXT INDEX Statement [page 1445]

1.4.4.278 TRUNCATE TEXT INDEX Statement

Deletes the data in a MANUAL or an AUTO REFRESH text index.

Syntax

```
TRUNCATE TEXT INDEX text-index-name
ON [ owner.]table-name
```

Parameters

ON clause

The name of the table on which the text index is built.

Remarks

Use the TRUNCATE TEXT INDEX statement when you want to delete data from a manual text index without dropping the text index definition. For example, to alter the text configuration object for the text index to change the stoplist, truncate the text index, change the text configuration object it refers to, and then refresh the text index to populate it with new data.

You cannot perform a TRUNCATE TEXT INDEX statement on a text index defined as IMMEDIATE REFRESH (the default). For IMMEDIATE REFRESH text indexes, you must drop the index instead.

The TRUNCATE TEXT INDEX requires exclusive access to the table. Any open cursors that reference the table being truncated must be closed, and a COMMIT or ROLLBACK statement must be executed to release the reference to the table.

Privileges

You must be the owner of the table, or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY TABLE system privilege

Side Effects

Automatic commit

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The first statement creates the txt_index_manual text index. The second statement populates the text index with data. The third statement truncates the text index data.

```
CREATE TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation ( Description )
   MANUAL REFRESH;
REFRESH TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation;
TRUNCATE TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation;
```

The truncated text index is repopulated with data the next time it is refreshed.

Related Information

Full Text Search
Text Index Concepts and Reference
CREATE TEXT INDEX Statement [page 1030]
ALTER TEXT INDEX Statement [page 762]
DROP TEXT INDEX Statement [page 1136]
REFRESH TEXT INDEX Statement [page 1325]

1.4.4.279 TRY Statement

Implements error handling for compound statements (if an error occurs in the TRY block, it passes control to another group of statements that is enclosed in a CATCH block).

Syntax

```
[ statement-label : ]
BEGIN TRY
   [ local-declaration; ... ]
   [ statement-list ]
END TRY
BEGIN CATCH
   [ statement-list ]
END CATCH
```

```
local-declaration :
variable-declaration
| cursor-declaration
```

```
| exception-declaration
| temporary-table-declaration

 : a valid DECLARE statement
```

Parameters

statement-label

When an ending `statement-label` is specified, it must match the beginning `statement-label`. The LEAVE statement can be used to resume execution at the first statement after the compound statement. The compound statement that is the body of a procedure or trigger has an implicit label that is the same as the name of the procedure or trigger.

local-declaration

Immediately following the BEGIN TRY, a compound statement can have local declarations for objects that only exist within the compound statement. A compound statement can have a local declaration for a variable, a cursor, a temporary table, or an exception. Local declarations can be referenced by any statement in that compound statement, or in any compound statement nested within it. Local declarations of the compound statement are visible to the exception handler for the statement. Local declarations are not visible to other procedures that are called from within a compound statement.

variable-declaration

A valid DECLARE statement.

exception-declaration

A valid DECLARE statement.

cursor-declaration

A valid DECLARE CURSOR statement.

temporary-table-declaration

A valid DECLARE LOCAL TEMPORARY TABLE statement.

Remarks

The CATCH block is the error handler for the TRY statement.

TRY...CATCH statements can be nested and used anywhere that a BEGIN...END statement can be used. Statements within the TRY block ignore the on_tsql_error and continue_after_raiserror database options, as well as the ON EXCEPTION RESUME clause of the CREATE PROCEDURE statement. TRY...CATCH statements are not atomic.

If no errors occur in the TRY block, the CATCH block is skipped and control passes to the statement following the CATCH block or the caller if no such statement exists. If an error occurs in one of the statements in the TRY block, control passes to the first statement in the CATCH block. Once the CATCH block completes, control passes to the statement following the CATCH block or the caller if no such statement exists. The effect of statements that precede a statement that generates an error is not reverted unless the exception handler

generates an error and is nested within an atomic block or an explicit ROLLBACK statement is called. In this case, all statements within the atomic transaction block are reverted.

Errors in the CATCH block are handled according to the connection and procedure settings unless the statements generating them are enclosed in additional TRY...CATCH statements.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

These examples use the following table:

```
CREATE TABLE t( col1 DOUBLE );
```

Executing the following compound statement inserts value 6 into table t:

```
BEGIN TRY
    DECLARE val INT;
    SET val = 0;
    INSERT INTO t VALUES( 1 / val );
    -- This statement will not be executed
    INSERT INTO t VALUES( val );
END TRY
BEGIN CATCH
    SET val = 6;
    INSERT INTO t VALUES( val );
END CATCH;
```

Executing the following procedure by using `CALL proc1(10);` inserts the following values into the table t:

-10

10

```
CREATE PROCEDURE DBA.proc1( v INTEGER )
BEGIN TRY
    DECLARE local_val INTEGER = 0;
    INSERT INTO t VALUES(-v);
    SET local_val = v / local_val;
    -- This statement will not be executed
    MESSAGE 'The value is ', v;
END TRY
BEGIN CATCH
    INSERT INTO t VALUES(v);
END CATCH;
```

Related Information

Exception Handling and Nested Compound Statements
Stored Procedures, Triggers, Batches, and User-defined Functions
Error and Warning Handling
Exception Handling and Atomic Compound Statements
Exception Handlers
Atomic Compound Statements
DECLARE Statement [page 1057]
DECLARE CURSOR Statement [ESQL] [SP] [page 1061]
DECLARE LOCAL TEMPORARY TABLE Statement [page 1066]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
CONTINUE Statement [page 818]
SIGNAL Statement [SP] [page 1405]
RESIGNAL Statement [SP] [page 1338]
RAISERROR Statement [page 1315]
BEGIN Statement [T-SQL] [page 787]

1.4.4.280 UNION Statement

Combines the results of two or more SELECT statements or query expressions.

Syntax

```
[ WITH temporary-views ] query-block
  UNION [ ALL | DISTINCT ] query-block
[ ORDER BY [ integer | select-list-expression-name ] [ ASC | DESC ], ... ]
[ FOR XML xml-mode ]
[ OPTION( query-hint, ... ) ]
```

```
temporary-views :
  regular-view, ...
| RECURSIVE { regular-view | recursive-view }, ...
```

```
regular-view :
  view-name [ ( column-name, ... ) ]
  AS ( query-block )
```

```
recursive-view :
  view-name ( column-name, ... )
  AS ( initial-query-block UNION ALL recursive-query-block )
```

```
query-block : see the documentation on common elements in SQL syntax
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| option-name = option-value
```

```
option-name : identifier
```

```
option-value :
hostvar (indicator allowed)
| string
| identifier
| number
```

Parameters

WITH or WITH RECURSIVE clause

Define one or more common table expressions, also known as temporary views, to be used elsewhere in the remainder of the statement. These expressions may be non-recursive, or may be self-recursive. Recursive common table expressions may appear alone, or intermixed with non-recursive table expressions, only if the RECURSIVE keyword is specified. Mutually recursive common table expressions are not supported.

This clause is permitted only if the SELECT query block appears in one of the following locations:

- Within a top-level SELECT query block including the top-level SELECT query block of a view definition
- Within a top-level SELECT statement within an INSERT query block
- Within a nested SELECT query block defining a derived table in any type of SQL statement

Recursive expressions consist of an initial subquery and a recursive subquery. The initial-query implicitly defines the schema of the view. The recursive subquery must contain a reference to the view within the FROM clause. During each iteration, this reference refers only to the rows added to the view in the previous iteration. The reference must not appear on the null-supplying side of an outer join. A recursive common table expression must not use aggregate functions and must not contain a GROUP BY, ORDER BY, or DISTINCT clause.

The WITH clause is not supported with remote tables. The WITH clause may also be used in INTERSECT, UNION, and EXCEPT query blocks.

This functionality is available only in the Watcom SQL dialect.

```
WITH RECURSIVE
  manager ( EmployeeID, ManagerID,
            GivenName, Surname, mgmt_level ) AS
( ( SELECT EmployeeID, ManagerID,       -- initial subquery
           GivenName, Surname, 0
    FROM Employees AS e
    WHERE ManagerID = EmployeeID )
  UNION ALL
  ( SELECT e.EmployeeID, e.ManagerID,   -- recursive subquery
           e.GivenName, e.Surname, m.mgmt_level + 1
    FROM Employees AS e JOIN manager AS m
     ON   e.ManagerID =  m.EmployeeID
      AND e.ManagerID <> e.EmployeeID
      AND m.mgmt_level < 20 ) )
SELECT 'Manager', * FROM manager
WHERE mgmt_level > 0
UNION ALL
SELECT 'Employee', * FROM manager
WHERE mgmt_level = 0
ORDER BY mgmt_level, Surname, GivenName;
```

ORDER BY clause

Specifies the final ordering of the results. The UNION ORDER BY clause uses column names as determined by the first `query-block`, however table references from the `query-block` may not be used in the ORDER BY clause.

For example, if the following statement was run without AS LastName, then specifying ORDER BY Surname would work, but ORDER BY t1.Surname would not:

```
SELECT t1.Surname AS LastName
FROM GROUPO.Employees t1
UNION
SELECT t1.Surname
FROM GROUPO.Customers t1
ORDER BY LastName;
```

FOR XML clause

The FOR XML clause is documented with the SELECT statement.

OPTION clause

Specifies hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- `option-name` = `option-value`.
  Use an `OPTION( isolation_level = ... )` specification in the query text to override all other means of specifying isolation level for a query.

Remarks

UNION ALL concatenates the results of the two query blocks into a single (larger) result set. Each query block may be nested. UNION DISTINCT eliminates duplicate rows in the final result. Eliminating duplicates requires extra processing, so UNION ALL should be used instead of UNION where possible. UNION DISTINCT is identical to UNION.

The result sets of the two `query-block`s must be UNION-compatible; they must each have the same number of items in their respective SELECT lists, and the types of each expression should be comparable. If corresponding items in two SELECT lists have different data types, the database server chooses a data type for the corresponding column in the result and automatically convert the columns in each `query-block` appropriately.

The column names displayed are the same column names that are displayed for the first `query-block` and these names are used to determine the expression names to be matched with the ORDER BY clause. An alternative way of customizing result set column names is to use a common table expression (the WITH clause).

Privileges

You must be the owner of the objects mentioned in `query-block`, or have SELECT privilege on the objects you are joining in the union.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Core Feature. Specifying the DISTINCT keyword with UNION is optional ANSI/ISO SQL Language Feature T551. Specifying an ORDER BY clause with UNION is ANSI/ISO SQL Language Feature F850. A `query-`

block that contains an ORDER BY clause constitutes ANSI/ISO SQL Feature F851. A query block that contains a row-limit clause (SELECT TOP or LIMIT) comprises optional ANSI/ISO SQL Language Feature F857 or F858, depending on the context. The FOR XML and OPTION clauses are not in the standard.

Transact-SQL

UNION and UNION ALL are supported by Adaptive Server Enterprise. The FOR XML and OPTION clauses are not supported.

Example

List all distinct surnames of employees and customers.

```
SELECT Surname
FROM GROUPO.Employees
UNION
SELECT Surname
FROM GROUPO.Customers;
```

Related Information

Common Elements in SQL Syntax [page 639]
Set Operators and the NULL Value
SELECT Statement [page 1362]
EXCEPT Statement [page 1148]
INTERSECT Statement [page 1242]

1.4.4.281 UNLOAD Statement

Unloads data from a data source into a file.

Syntax

```
UNLOAD data-source
{ TO filename
   | INTO FILE filename
   | INTO CLIENT FILE client-filename
   | INTO VARIABLE variable-name }
[ unload-option ... ]
```

```
data-source :
[ FROM ] [ TABLE ] [ owner.]table-name
| [ FROM ] [ MATERIALIZED VIEW ] [ owner.]materialized-view-name
| select-statement
```

```
filename : string | variable
```

```
client-filename : string | variable
```

```
unload-option :
APPEND { ON | OFF }
| BYTE ORDER MARK { ON | OFF }
| { COMPRESSED | NOT COMPRESSED }
| DELIMITED BY string
| ENCODING encoding
| { ENCRYPTED KEY 'key' [ ALGORITHM 'algorithm' ] | NOT ENCRYPTED }
| ESCAPE CHARACTER character
| ESCAPES { ON | OFF }
| FORMAT { TEXT | BCP }
| HEXADECIMAL { ON | OFF }
| ORDER { ON | OFF }
| QUOTE string
| QUOTES { ON | OFF | ALL }
| ROW DELIMITED BY string
| WITH COLUMN NAMES
```

```
encoding : string
```

```
algorithm :
'AES' | 'AES256' | 'AES_FIPS' | 'AES256_FIPS'
```

Parameters

TO clause

The name of the file to unload data into. The `filename` path is relative to the database server's starting directory. If the file does not exist, it is created. If it already exists, it is overwritten unless APPEND ON is also specified.

INTO FILE clause

Semantically equivalent to TO `filename`.

INTO CLIENT FILE clause

The file on the client computer into which the data is unloaded. If the file doesn't exist, it is created. If it already exists, it is overwritten unless APPEND ON is also specified. The path is resolved on the client computer relative to the current working directory of the client application.

INTO VARIABLE clause

The variable to unload the data into. The variable must already exist and be of CHAR, NCHAR or BINARY type. The APPEND option causes the unloaded data to be concatenated to the current contents of the variable.

APPEND clause

When APPEND is ON, unloaded data is appended to the end of the file specified. When APPEND is OFF, unloaded data replaces the contents of the file specified. APPEND is OFF by default. This clause cannot be specified when specifying the COMPRESSED or ENCRYPTED clauses, and cannot be used if the file being appended to is compressed or encrypted.

BYTE ORDER MARK clause

Use this clause to specify whether a byte order mark (BOM) should be written. By default, this option is ON, provided the destination for the unload is a local or client file. When the BYTE ORDER MARK option is ON and the ENCODING is UTF-8 or UTF-16, then a BOM is written. If BYTE ORDER MARK is OFF, a BOM is not unloaded.

COMPRESSED clause

Specifies whether to compress the data. The default is NOT COMPRESSED. You cannot compress the data if you want the data to be appended (APPEND ON).

DELIMITED BY clause

The string used between columns. The default column delimiter is a comma. Specify an alternative column delimiter by providing a string up to 255 bytes in length.

ENCODING clause

All database data is translated from the database character encoding to the specified CHAR or NCHAR encoding. When ENCODING is not specified, the database's CHAR encoding is used.

If a translation error occurs during the unload operation, it is reported based on the setting of the on_charset_conversion_failure option.

Specify the BYTE ORDER MARK clause to include a byte order mark in the data.

ENCRYPTED clause

Specifies whether to encrypt the data. If you specify NOT ENCRYPTED (the default), the data is not encrypted. If you specify ENCRYPTED KEY with a key and no algorithm, the data is encrypted using AES128 and the specified key. The key can be either a string or a variable name. If you specify ENCRYPTED KEY with a key and algorithm, the data is encrypted using the specified key and algorithm. The algorithm can be any of the algorithms accepted by the CREATE DATABASE statement. You cannot specify the SIMPLE obfuscation algorithm.

You cannot encrypt the data if you want the data to be appended (APPEND ON).

ESCAPES clause

With ESCAPES turned ON (the default), the database server writes escape sequences. Newline characters can be written as the combination \n, other characters can be included in data as hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\ ) is written as a single backslash. A backslash followed by any character other than n, x, X, or \ is written as two separate characters. For example, \q inserts a backslash and the letter q. It is recommended that the string you specify for the escape character is no longer than one multibyte character.

FORMAT clause

Outputs data in either TEXT format or in BCP out format. If you choose TEXT, output lines are written as text characters, one row per line, with values separated by the column delimiter string. If you choose BCP, data including BLOBs are exported as BCP input files for use with Adaptive Server Enterprise. The default format is TEXT.

HEXADECIMAL clause

By default, HEXADECIMAL is ON. Binary column values are written as 0x*nnnnnn*..., where 0x is a zero followed by an x, and each *n* is a hexadecimal digit. It is important to use HEXADECIMAL ON when dealing with multibyte character sets.

The HEXADECIMAL clause can be used only with the FORMAT TEXT clause.

ESCAPE CHARACTER clause

Use this clause to specify the escape character used in the data. The default escape character for characters written as hexadecimal codes and symbols is a backslash (\), so \x0A is the line feed character, for example. This can be changed using the ESCAPE CHARACTER clause.

It is recommended that the string you specify for the escape character is no longer than one multibyte character.

ORDER clause

With ORDER ON (the default), the exported data is ordered by clustered index if one exists. If a clustered index does not exist, the exported data is ordered by primary key values. With ORDER OFF, the data is exported in the same order you see when selecting from the table without an ORDER BY clause. Exporting is slower with ORDER ON. However, reloading using the LOAD TABLE statement is quicker because of the simplicity of the indexing step.

For UNLOAD `select-statement`, the ORDER clause is ignored. However, you can still order the data by specifying an ORDER BY clause in the SELECT statement.

QUOTE clause

The QUOTE clause is for TEXT data only; the `string` is placed around string values. The default is a single apostrophe ('). When using QUOTES ALL, the quote character can only be a single apostrophe (') or quotation mark (").

QUOTES clause

With QUOTES ON (the default), the quote character, which defaults to a single quote (apostrophe), is placed around all exported strings. If QUOTES ALL is specified, the quote string is placed around all values, not just around strings. To suppress quoting, use QUOTES OFF.

ROW DELIMITED BY clause

Use this clause to specify the string that indicates the end of a record. The default delimiter string is '\x0d \x0a' (CR/LF) for Microsoft Windows and '\x0a' (LF) for UNIX and Linux. However, it can be any string up to up to 255 bytes in length; for example, `ROW DELIMITED BY '###'`. To specify tab-delimited rows, you could specify the hexadecimal escape sequence for the tab character (ordinal value 9), `ROW DELIMITED BY '\x09'`. If your delimiter string contains a newline character (\n), it results in a line feed (LF) character only. On Microsoft Windows systems, it is customary to use '\x0d\x0a' to write out carriage return followed by line feed (CR/LF) to text files.

WITH COLUMN NAMES clause

The WITH COLUMN NAMES clause inserts the column names in the first line of the file or variable.

Remarks

The UNLOAD `select-statement` statement allows data from a SELECT statement to be exported to a comma-delimited file. The result set is not ordered unless the SELECT statement contains an ORDER BY clause.

The UNLOAD TABLE statement allows efficient mass exporting from a database table or materialized view into a file. The UNLOAD TABLE statement is more efficient than the Interactive SQL statement OUTPUT, and can be called from any client application.

The database server, or the client application, depending upon whether INTO FILE or INTO CLIENT FILE was specified, respectively, must have operating system permissions to write to the specified file. INTO CLIENT FILE is not supported for **Tabular Data Stream** (**TDS**) connections.

When unloading table columns with binary data types, UNLOAD TABLE writes hexadecimal strings, of the form **\x** nnnn, where n is a hexadecimal digit.

When unloading and reloading a database that has proxy tables, you must create an external login to map the local user to the remote user, even if the user has the same password on both the local and remote databases. If you do not have an external login, the reload may fail because you cannot connect to the remote server.

When unloading into a variable (INTO VARIABLE), the output is converted to a character set as follows:

CHAR

write to the variable in CHAR encoding. The ENCODING clause must match the CHAR encoding.

NCHAR

write to the variable in NCHAR encoding. The ENCODING clause must match the NCHAR encoding.

BINARY

write to the variable in BINARY encoding. The ENCODING clause must match the BINARY encoding; otherwise, the CHAR encoding is used.

If you choose to compress and encrypt the unloaded data, it is compressed first.

UNLOAD TABLE places an exclusive lock on the whole table or materialized view.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

When the UNLOAD statement is executed, the @@rowcount variable is set to the number of rows unloaded.

To retain maximum precision of date values, set the date_format to YYYY-MM-DD.

To retain maximum precision of TIMESTAMP values, set the timestamp_format to YYYY-MM-DD HH:NN:SS.SSSSSS.

To retain maximum precision of TIMESTAMP WITH TIME ZONE values, set the timestamp_with_time_zone_format to YYYY-MM-DD HH:NN:SS.SSSSSS+HH:NN.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

Privileges

When unloading into a variable, no privileges are required. Otherwise, the required privileges depend on the database server -gl option, as follows:

- If the -gl option is set to ALL, you must be the owner of the tables, or have SELECT privilege on the tables, or have the SELECT ANY TABLE system privilege.
- If the -gl option is set to DBA, you must have the SELECT ANY TABLE system privilege.
- If the -gl option is set to NONE, UNLOAD is not permitted.

When unloading to a file on a client computer:

- You must have the WRITE CLIENT FILE privilege.

- You must have write permissions on the directory where the file is located.
- The allow_write_client_file database option must be enabled.
- The WRITE_CLIENT_FILE feature must be enabled.

Side Effects

None. The query is executed at the current isolation level.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example unloads the contents of the Products table to a UTF-8-encoded file, `productsT.dat`:

```
UNLOAD TABLE GROUPO.Products TO 'c:\\temp\\productsT.dat' ENCODING 'UTF-8';
```

The following example creates a variable called @myProducts and then unloads the Products.Name column into the variable:

```
CREATE VARIABLE @myProducts LONG VARCHAR;
UNLOAD SELECT NAME FROM GROUPO.Products INTO VARIABLE @myProducts ESCAPE
CHARACTER '!';
```

Related Information

Clustered Indexes
Supported Character Sets
Tips on Exporting Data with the UNLOAD Statement
Access to Data on Client Computers
Data Import and Export
Tips on Exporting Data with the UNLOAD TABLE Statement
CREATE DATABASE Statement [page 821]
LOAD TABLE Statement [page 1247]
PASSTHROUGH Statement [SQL Remote] [page 1299]
OUTPUT Statement [Interactive SQL] [page 1289]
-sf Database Server Option

allow_write_client_file Option
-gl Database Server Option
timestamp_format Option
date_format Option
on_charset_conversion_failure Option
progress_messages Option

1.4.4.282 UNPIVOT Clause

Unpivots compatible-type columns of a table expression in a FROM clause (FROM `unpivoted-derived-table expression`) into rows in a derived table. Unpivoting is used to normalize data, for example when you have similar data stored in multiple columns in tables and you want to return it in one column.

Syntax

```
FROM unpivoted-derived-table
unpivoted-derived-table :
unpivot-source-table UNPIVOT [ { INCLUDE | EXCLUDE } NULLS ]  ( unpivot-
clause ) [ AS ] correlation-name
unpivot-clause :
unpivot-value-clause unpivot-for-clause unpivot-in-clause
unpivot-value-clause :
value-column
| ( value-column [,...] )
unpivot-for-clause :
FOR unpivot-column
unpivot-in-clause :
IN ( unpivot-old-column [[ AS ] unpivot-old-column-alias ] [,...] )
| IN (( unpivot-old-column [,...] ) [[ AS ] unpivot-old-column-alias ] [,...] )
```

Parameters

{ INCLUDE | EXCLUDE } NULLS

Specify whether to include or exclude NULL values in the results for `value-column`.The default behavior is EXCLUDE NULLS.

unpivot-value-clause

Specify name(s) for the new columns in the unpivoted derived table that will hold unpivoted values.

unpivot-for-clause

Specify a column to unpivot the data for.

unpivot-in-clause

Specify the values of `unpivot-for-clause` to unpivot data for.

Remarks

An unpivoted derived table contains a subset of the columns in `unpivot-source-table`, plus additional columns specified in the `unpivot-value-clause` and `unpivot-column`. The columns in the IN clause must be found in `unpivot-source-table` but are not present in the unpivoted derived table.

If the IN clause has `I` elements, then each row in `unpivot-source-table` is transformed into `I` rows in the unpivoted derived table. That is, each value tuple corresponding to an item in the IN clause generates a new row in the unpivoted derived table. The `unpivot-column` value for each row is set to the value of the IN clause alias for that item, while the new columns specified in `unpivot-value-clause` are set to the value tuple of the original row for that item.

Each item in the IN clause requires compatible domains with any other items in the IN clause. The data type for the new columns specified in `unpivot-value-clause` corresponds to a super type of these compatible domains. The data type of the values in `unpivot-column` is NCHAR, and its values are the aliases defined in the IN clause.

Privileges

You must have SELECT privileges on the objects referenced in `unpivot-source-table`.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

Suppose you have a table called MyCustomers, containing the names of your contacts and the various phone numbers they have. The following statements create such a table, populate it with fictitious data, and then query the table:

```
CREATE TABLE MyCustomers
( ContactID INT PRIMARY KEY,
  LastName VARCHAR(64),
  FirstName VARCHAR(64),
  Home VARCHAR(32),
```

```
  Mobile VARCHAR(32),
  AltPhone VARCHAR(32)
);
INSERT MyCustomers
  ( ContactID, LastName, FirstName, Home, Mobile, AltPhone )
VALUES
  ( 1, 'Bringle', 'Susan', '111-593-1451', '222-693-7620', NULL ),
  ( 2, 'Hoffsteter', 'Garth', '113-249-6622', NULL, NULL),
  ( 3, 'Zenibar', 'Austin', NULL, '171-765-8730', '888-536-5324' );
SELECT * FROM MyCustomers;
```

ContactID	LastName	FirstName	Home	Mobile	AltPhone
1	Bringle	Susan	111-593-1451	222-693-7620	(NULL)
2	Hoffsteter	Garth	113-249-6622	(NULL)	(NULL)
3	Zenibar	Austin	(NULL)	171-765-8730	888-555-5555

For each contact, the contact numbers are stored in three separate columns: Home, Mobile, and AltPhone.

Now, suppose that you need to gather some statistics related to phone numbers that your customer service representatives call. You want to see a list of all phone numbers that are called, with the numbers sorted so that you can identify trends such as similar exchange numbers. To get the results you need, you must unpivot the Home, Mobile, and AltPhone columns into a single column that contains one row per phone number, sorted by phone number. Your SELECT statement might look as follows:

```
SELECT ContactID, PhoneNumber, PhoneType
FROM
(
  SELECT ContactID, Home, Mobile, AltPhone
  FROM MyCustomers
) AS MyUnpivotSource
UNPIVOT EXCLUDE NULLS
(
  PhoneNumber FOR PhoneType IN ( Home, Mobile, AltPhone )
)
AS MyUnpivotedTable
ORDER BY PhoneNumber;
```

ContactID	PhoneNumber	PhoneType
1	111-593-1451	Home
2	113-249-6622	Home
3	171-765-873	Mobile
1	222-693-7620	Mobile
3	888-536-5324	AltPhone

The results show how the UNPIVOT operation normalizes the three columns of phone number data found in the MyCustomers table, placing them into a single phone number column called PhoneNumber, which is `unpivot-value-clause` in this example. The values in the PhoneType column, which is `unpivot-column` in this example, are determined by the column in which the phone number value was found in `unpivot-source-table`.

The following query extracts the phone numbers for contacts and their customers in the Customers table into an unpivoted derived table where the phones are listed, and the provenance - customer phone or contact phone - is recorded in the unpivot column People.

```
SELECT DISTINCT *
FROM
 ( SELECT CO.CustomerID,
          C.City customer_city,
          C.State customer_state,
          c.Phone customer_phone,
          CO.City contact_city,
          CO.State contact_state,
          CO.Phone contact_phone
   FROM Customers C
   JOIN Contacts CO
   WHERE  C.State IN ( 'NJ', 'NY' ) ) UnpivotSourceTable
UNPIVOT
 ( ( City, State, Phone ) FOR People IN
     (
       ( customer_city, customer_state, customer_phone ) AS CustomerPhone,
       ( contact_city, contact_state, contact_phone ) AS ContactPhone
     )
 ) UnpivotedDerivedTable
ORDER BY CustomerID, People;
```

For example, the unpivot source table UnpivotSourceTable contains the following row (among others):

CustomerID	customer_city	customer_state	customer_phone	contact_city	contact_state	contact_phone
101	Kingston	NJ	2015558966	Hespeler	NJ	6035550988

The query unpivots the data into new rows identified by CustomerID 101 in the result set table below.

CustomerID	People	City	State	Phone
101	ContactPhone	Hespeler	NJ	6035550988
101	CustomerPhone	Kingston	NJ	2015558966
111	ContactPhone	Cornwall	NY	6175557956
111	ContactPhone	Cornwall	NY	6175558890
111	CustomerPhone	Hastings	NY	3155554486
116	ContactPhone	Cornwall	NY	6175552222
116	ContactPhone	Cornwall	NY	6175559877
116	CustomerPhone	Stayner	NY	9145553817
147	ContactPhone	Hespeler	NJ	6035551200
147	CustomerPhone	Campbellford	NJ	9085556021

Related Information

Tutorial: Pivoting Table Data
FROM Clause [page 1173]
PIVOT Clause [page 1300]

1.4.4.283 UPDATE Statement

Modifies rows in database tables.

Syntax

Update rows using joins, views, and derived tables

```
UPDATE [ row-limitation ] table-expression [, ...]
SET set-item[, ...]
[ FROM table-expression [, ...] ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ] , ...]
[ OPTION( query-hint, ... ) ]
```

```
row-limitation :
FIRST
| TOP { ALL | limit-expression } [ START AT startat-expression ]
limit-expression : simple-expression
startat-expression : simple-expression
```

```
simple-expression :
integer
| variable
| ( simple-expression )
| ( simple-expression { + | - | * } simple-expression )
```

```
table-expression : A table expression that can include joins, outer joins,
views and derived tables. See the FROM clause topic.
```

```
set-item :
[ correlation-name.]column-name={ expression | DEFAULT }
| [ owner. ]table-name.column-name = { expression | DEFAULT }
| [ owner. ]@variable-name=expression
```

```
table-name :
[ owner.]base-table-name
| temporary-table-name
| derived-table-name
| [ owner.]view-name
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| FORCE NO OPTIMIZATION
| option-name = option-value
```

```
option-name : identifier
```

```
option-value :
hostvar (indicator allowed)
| string
| identifier
| number
```

Update rows in a single table

```
UPDATE [ row-limitation ] table-name
```

```
SET set-item[, ...]
[ FROM table-expression [, ...] ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ] , ...]
[ OPTION( query-hint, ... ) ]
```

```
table-name :
[ owner.]table-name [ [ AS ] correlation-name ]
| [ owner.]view-name [ [ AS ] correlation-name ]
| derived-table
```

```
derived-table :
( select-statement )
[ AS ] correlation-name [ ( column-name [, ...] ) ]
```

Parameters

UPDATE clause

When updating tables, `table-expression` can include temporary tables, global temporary tables, derived tables, or views. Views and derived tables can be updated unless they are non-updatable.

When updating using joins, views, and derived tables, a list of more than one `table-expression` results in a Cartesian product of the rows formed by the underlying table expressions, which can then be restricted via the use of the WHERE clause.

UPDATES can be performed on views only if the query specification defining the view is updatable.

row-limitation clause

The row limitation clause restricts the rows being updated to only a subset of the rows that satisfy the WHERE clause. The TOP and START AT arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The TOP argument must evaluate to a value greater than or equal to 0. The START AT argument must evaluate to a value greater than 0. An ORDER BY clause should be used to order the rows in a meaningful manner.

SET clause

Use the SET clause to set column names or variables to the specified expression.

Use the SET clause to set the column to a computed column value by using this format:

```
SET column-name = expression, ...
```

Each specified column is set to the value of the expression. There are no restrictions on `expression`. If `expression` is a `column-name`, then the previous value from that column is used.

If a column has a default defined, then use the SET clause to set a column to its default value.

You can also use the SET clause to assign a variable by using the following format:

```
SET @variable-name = expression, ...
```

The `owner` specification is only for use with database-scope variables.

When assigning a value to a variable, the variable must already be declared, and its name must begin with the at sign (@). If the variable name matches the name of a column in the table to be updated, then the UPDATE statement updates the column value and leaves the variable unchanged. Variable and column assignments can be combined in any order.

FROM clause

The FROM `table-expression` clause allows tables to be updated based on joins. `table-expression` can contain arbitrary complex table expressions, such as OUTER, CROSS, and NATURAL joins.

If the FROM clause is present, then `table-name` must specify the sole table to be updated, and it must qualify the name in the same way as it appears in the FROM clause. If correlation names are used in the FROM clause, then the identical correlation name must be specified as `table-name`. If the table expression to be updated is a derived table, then the derived table must be repeated in the `table-name` specification.

Updating a specified table is not allowed if the ansi_update_constraints option is set to Strict.

If a FROM clause is specified, then the SET clause can specify only columns from `table-name` to be updated. Otherwise, an error is generated.

The following statement illustrates a potential ambiguity in table names in UPDATE statements using the syntax for updating a specific table that uses correlation names:

```
UPDATE table_1
SET column_1 = ...
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

In the following example, each instance of table_1 in the FROM clause has a correlation name, denoting a self-join of table_1 to itself. However, the UPDATE statement fails to specify which of the rows that make up the self-join are to be updated. This omission can be corrected by specifying the correlation name in the UPDATE statement as follows:

```
UPDATE table_1 AS alias_1
SET column_1 = ...
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

WHERE clause

If a WHERE clause is specified, then only rows satisfying the search condition are updated. If no WHERE clause is specified, then every row is updated.

ORDER BY clause

Normally, the order in which rows are updated does not matter. However, with the FIRST or TOP clause, the order can be significant.

You cannot use ordinal column numbers in the ORDER BY clause.

To use the ORDER BY clause, the ansi_update_constraints option must not be set to Strict.

To update columns that appear in the ORDER BY clause, the ansi_update_constraints option must be set to Off.

OPTION clause

Use this clause to specify hints for executing the statement. The setting you specify is only applicable to the current statement and takes precedence over any public or temporary option settings, including those set by ODBC-enabled applications. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- `option-name` = `option-value`.
  Use an `OPTION( isolation_level = ... )` specification in the query text to override all other means of specifying isolation level for a query.
  Use an `OPTION( parameterization_level = ... )` specification in the query text to override the parameterization level for a query.

Remarks

The UPDATE statement is used to modify the rows in one or more tables. Each named column is set to the value of the expression on the right side of the equal sign. There are no restrictions on the `expression`. Even `column-name` can be used in the expression. The old value is used.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive or not. A CHAR data type column updated with the string Street is always held in the database with an uppercase S and the remainder of the letters lowercase. SELECT statements return the string as Street. If the database is not case sensitive, however, then all comparisons make Street the same as street, STREET, and so on. Further, if a single-column primary key already contains an entry Street, then an UPDATE of another row's primary key to street is rejected, as it would make the primary key not unique.

If the new value does not differ from the old value, then no change is made to the data. However, BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whether the new value differs from the old value. AFTER UPDATE triggers fire only if the new value is different from the old value.

Updating a significant amount of data using the UPDATE statement also updates column statistics.

If no WHERE clause is specified, then every row is updated. If a WHERE clause is specified, then only those rows which satisfy the search condition are updated.

Normally, the order that rows are updated in does not matter. However, with the NUMBER(*) function, an ordering can be useful to get increasing numbers added to the rows in some specified order. Also, to do something like add 1 to the primary key values of a table, it is necessary to do this in descending order by primary key, so that you do not get duplicate primary keys during the operation.

Views can be updated provided the SELECT statement defining the view does not contain a GROUP BY clause, an aggregate function, or involve a UNION clause.

The optional FROM clause allows tables to be updated based on joins. If the FROM clause is present, then the WHERE clause qualifies the rows of the FROM clause. When specifying a FROM clause, qualify the table name that is being updated the same way in both parts of the statement. If a correlation name is used in one place, then use the same correlation name in the other. Otherwise, an error is generated.

When updating database-scope variables using the SET clause, the setting does not persist between restarts of the database, even though the variable does. When a database is restarted, the value of a database-scope

variable reverts to NULL or its default, if defined. The SYSDATABASEVARIABLE system view contains a list of all database-scope variables and their default values.

You cannot update a database-scope variable owned by another user.

Privileges

You must be the owner of the table being updated, or have UPDATE privilege on the columns being modified, or have the UPDATE ANY TABLE system privilege.

No privileges are required to update a self-owned database-scope variable. To update a database-scope variable owned by PUBLIC, you must have the UPDATE PUBLIC DATABASE VARIABLE system privilege.

Side Effects

Column statistics are updated to reflect the modified values.

If a table has a primary key, a UNIQUE constraint, or a UNIQUE index, then the processing of the UPDATE statement may involve the use of a temporary table if the table manipulations cannot be performed without violating the uniqueness constraint. The temporary table stores rows modified by the UPDATE statement that violate one or more uniqueness constraints. These rows are temporarily deleted from the base table during the execution of the UPDATE statement, and are subsequently re-inserted. This behavior may have implications for AFTER triggers and other concurrent connections.

Standards

ANSI/ISO SQL Standard

The UPDATE...`table-expression` syntax is a Core Feature of the ANSI/ISO SQL Standard. However the following extensions are not in the standard:

- The FROM and ORDER BY clauses.
- The `row-limitation` clause.
- The ability to specify more than one `table-expression`.
- The ability to update a variable using the SET clause.
- The OPTION clause.

The UPDATE...`table-expression` syntax also includes support for two optional ANSI/ISO SQL Language Features:

- Support for updating a join, possibly including one or more derived tables, comprises part of optional ANSI/ISO SQL Language Feature T111, "Updatable joins, unions, and columns".
- Support for modifying a table referenced in a nested subquery that forms part of the search condition for the UPDATE statement comprises optional ANSI/ISO Language Feature F781, "Self-referencing operations".

With the UPDATE...`table-expression` syntax, the setting of the ansi_update_constraints option controls which forms of table expressions can be modified. To enforce ANSI/ISO Core Feature compatibility, ensure that the ansi_update_constraints option is set to Strict.

The UPDATE...`table-name` syntax is not in the standard.

Example

Using the sample database, this example transfers employee Philip Chin (employee 129) from the sales department to the marketing department.

```
UPDATE GROUPO.Employees
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

Using the sample database, this example renumbers all existing sales orders by subtracting 2000 from the ID.

```
UPDATE GROUPO.SalesOrders AS orders
SET orders.ID = orders.ID - 2000
ORDER BY orders.ID ASC;
```

This update is possible only if the foreign key of the SalesOrderItems table (referencing the primary key SalesOrders.ID) is defined with the action ON UPDATE CASCADE. The SalesOrderItems table is then updated as well. Because the statement specifies an ORDER BY clause, and the ordering attribute is also specified in the SET clause, the ansi_update_constraints option must be set to Off or an error is returned.

Using the sample database, this example changes the price of a product at isolation level 2, rather than using the current isolation level setting of the database.

```
UPDATE GROUPO.Products
SET UnitPrice = 7.00
WHERE ID = 501
OPTION( isolation_level = 2 );
```

This example requires the ansi_update_constraints option to be set to a value other than Strict. Again using the sample database, this example uses the UPDATE...`table-name` syntax to reset the quantity-on-hand of those Tee Shirts where there exists at least one order whose quantity exceeds the current quantity-on-hand:

```
UPDATE GROUPO.Products AS a
SET Quantity = 0
FROM GROUPO.Products a JOIN GROUPO.SalesOrderItems b ON a.ID = b.ProductID
WHERE a.Name = 'Tee Shirt' AND b.Quantity > a.Quantity;
```

This example requires the ansi_update_constraints option to be set to a value other than Strict. In this example, the UPDATE...`table-expression` syntax is used to reset the quantity-on-hand for those Tee Shirts, and to reset the ShipDate for the Tee Shirt order to today's date:

```
UPDATE GROUPO.Products a JOIN GROUPO.SalesOrderItems b on a.ID = b.ProductID
SET a.Quantity = 0, b.ShipDate = CAST( NOW() AS DATE)
WHERE a.Name = 'Tee Shirt' AND b.Quantity > a.Quantity
```

This example shows how to update a table to set a column to its default value. In this example, you create a table, MyTable, populate it with data, and then execute an UPDATE statement specifying the SET clause to update some column values.

```
CREATE OR REPLACE TABLE MyTable(
   PK INT PRIMARY KEY  DEFAULT AUTOINCREMENT,
   TableName CHAR(128) NOT NULL,
   TableNameLen INT DEFAULT 20,
   LastUser CHAR(10) DEFAULT LAST USER,
   LastTime TIMESTAMP DEFAULT TIMESTAMP,
   LastTimestamp TIMESTAMP DEFAULT CURRENT TIMESTAMP );
INSERT INTO MyTable WITH AUTO NAME
   SELECT
      LENGTH(t.table_name) AS TableNameLen,
      t.table_name AS TableName
   FROM SYS.SYSTAB t
   WHERE table_id <= 10;
WAITFOR DELAY '00:00:05';
UPDATE MyTable
   SET TableName = TableName || '*',
       LastTimestamp = DEFAULT
   WHERE TableName LIKE '%idx%';
SELECT * FROM MyTable;
```

In this example, the LastTime column is automatically updated because its default is TIMESTAMP. Ordinarily, the LastTimestamp column would not update since its default is CURRENT TIMESTAMP. The SET statement forces an update of this column value using its default.

The following UPDATE statement does not use a FROM clause:

```
UPDATE Part
SET Description = "Bolt"
WHERE PartID = 9;
```

The following Transact-SQL example uses a FROM clause and allows you to update rows in a table from a more complex table expression. Part in the first clause should name a base table from the table expression specified in the FROM clause.

```
UPDATE Part
SET DESCRIPTION = "Bolt"
FROM Part JOIN Lineitem ON( Part.PartID = Lineitem.PartID)
WHERE Lineitem.quantity > 5;
```

Related Information

Locks During Updates
Data Changes Using UPDATE
FROM Clause [page 1173]
INSERT Statement [page 1232]
UPDATE Statement [SQL Remote] [page 1472]
UPDATE (Positioned) Statement [ESQL] [SP] [page 1470]
UPDATE Statements with a VERIFY Clause
UPDATE Statement Replication
ansi_update_constraints Option

1.4.4.284 UPDATE (Positioned) Statement [ESQL] [SP]

Modifies the data at the current location of a cursor.

Syntax

Embedded SQL

```
UPDATE WHERE CURRENT OF cursor-name
{ USING [ SQL ] DESCRIPTOR sqlda-name | { [ FROM ] | [ USING ] } hostvar-
list }
```

Stored procedures

```
UPDATE update-table, ...
SET set-item, ...
WHERE CURRENT OF cursor-name
```

```
hostvar-list : indicator variables allowed
```

```
update-table :
[owner-name.]object-name [ [ AS ] correlation-name ]
```

```
set-item :
[ correlation-name.]column-name = { expression | DEFAULT }
| [ owner-name. ]object-name.column-name = { expression | DEFAULT }
```

```
object-name : identifier (a table or view name)
```

```
sqlda-name : identifier
```

Parameters

UPDATE clause

Each `update-table` is matched to a table in the query for the cursor as follows:

1. If a correlation name is specified, it is matched to a table in the cursor's query that has the same `table-or-view-name` and the same `correlation-name`.
2. If there is a table in the cursor's query that has the same `table-or-view-name` that does not have a correlation name or has a correlation name that is the same as the `table-or-view-name`, then the update table is matched with this table in the cursor's query.
3. If there is a single table in the cursor's query that has the same `table-or-view-name` as the update table, then the update table is matched with this table in the cursor's query.

If a column has a default defined, use the SET clause to set a column to its default value.

USING DESCRIPTOR clause

When assigning a variable, the variable must already be declared, and its name must begin with the at sign (@). Variable and column assignments can be mixed together, and any number can be used.

SET clause

The columns in `set-item` must be in the table or view being updated. If a name on the left side of an assignment in the SET list matches a column in the updated table and the variable name, the statement updates the column.`set-item` cannot refer to aliases or columns from other tables or views. If the table or view you are updating is given a correlation name in the cursor specification, use the correlation name in the SET clause.

Each `set-item` is associated with a single `update-table`, and the corresponding column of the matching table in the cursor's query is modified. `expression` references columns of the tables identified in the UPDATE list and can use constants, host variables, variables, expressions from the SELECT list of the query, or combinations of the above using operators such as +, -, ..., COALESCE, IF, and so on. `expression` cannot reference aliases of expressions from the cursor's query or columns of other tables of the cursor's query that are not in the UPDATE list. Subselects, subquery predicates, and aggregate functions cannot be used in `set-item`.

Remarks

This form of the UPDATE statement updates the current row of the specified cursor. The current row is defined to be the last row successfully fetched from the cursor, and the last operation on the cursor must not have been a positioned DELETE statement.

In the Embedded SQL syntax, columns from the SQLDA or values from the host variable list correspond one-to-one with the columns returned from the specified cursor. If the sqldata pointer in the SQLDA is the null pointer, the corresponding SELECT list item is not updated.

In the stored procedure syntax, the requested columns are set to the specified values for the row at the current row of the specified query. The columns do not need to be in the SELECT list of the specified open cursor. This format can be prepared.

Also, when assigning a variable, the variable must already be declared, and its name must begin with the at sign (@). Variable and column assignments can be mixed together, and any number can be used. If a name on the left side of an assignment in the SET list matches a column in the updated table and the variable name, the statement updates the column.

The USING DESCRIPTOR, FROM `hostvar-list`, and `hostvar` formats are for Embedded SQL only.

Privileges

You must be the owner of the table being updated, or have UPDATE privilege on the columns being modified, or have the UPDATE ANY TABLE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The Embedded SQL syntax is not in the standard.

The stored procedure syntax is a Core Feature. If used within an Embedded SQL program, this syntax comprises part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The ability to specify more than one table to be updated is not in the standard.

The range of cursors that can be updated is dependent upon the setting of the ansi_update_constraints option. The ability to perform a positioned update over a cursor that is ordered (the SQL query has an ORDER BY clause) comprises optional ANSI/ISO SQL Language Feature F831, "Full cursor update". Performing a positioned update over more complex SQL constructions may involve additional extensions that are not in the standard.

Example

The following is an example of an UPDATE statement executed on a fictitious cursor called emp_cursor:

```
UPDATE GROUPO.Employees
SET Surname = 'Jones'
WHERE CURRENT OF emp_cursor;
```

Related Information

INSERT Statement [page 1232]
LOAD TABLE Statement [page 1247]
MERGE Statement [page 1271]
DELETE Statement [page 1070]
DELETE Statement (Positioned) [ESQL] [SP] [page 1074]
UPDATE Statement [page 1463]
ansi_update_constraints Option

1.4.4.285 UPDATE Statement [SQL Remote]

Modifies data in the database.

Syntax

Single-row updates of a single table executed by the Message Agent

```
UPDATE table-name
SET column-name = expression, ...
```

```
[ VERIFY ( column-name, ... ) VALUES ( expression, ... ) ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ], ... ]
[ OPTION( query-hint, ... ) ]
```

```
query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| FORCE NO OPTIMIZATION
| option-name = option-value
```

```
option-name : identifier
```

```
option-value :
hostvar (indicator allowed)
| string
| identifier
| number
```

Implement a specific SQL Remote feature

```
UPDATE table-name
PUBLICATION publication-name
{ SUBSCRIBE BY subscription-expression |
   OLD SUBSCRIBE BY old-subscription-expression
   NEW SUBSCRIBE BY new-subscription-expression }
WHERE search-condition
```

```
expression : value | subquery
```

Parameters

Syntax - Implement a specific SQL Remote feature

table-name

The `table-name` indicates the base table that must be modified on the remote databases.

publication-name

The `publication-name` indicates the publication for which subscriptions must be changed.

subscription-expression

The value of `subscription-expression` is used by SQL Remote to determine both new and existing recipients of the rows. The `subscription-expression` is either a value or a subquery. Alternatively, you can provide both OLD and NEW subscription expressions.

WHERE clause

The WHERE clause specifies which rows are to be transferred between subscribed databases.

Remarks

Syntax - Single-row updates of a single table executed by the Message Agent

The SQL Remote Message Agent uses this syntax when applying messages to the database. All of the parameters and remarks about updating rows on a single table in the standard UPDATE statement also apply to this variation of the UPDATE statement.

The VERIFY clause contains a set of values that are expected to be present in the row being updated. If the values do not match, any RESOLVE UPDATE triggers are fired before the UPDATE proceeds. The UPDATE does not fail if the VERIFY clause fails to match. When the VERIFY clause is specified, only one table can be updated at a time.

Syntax - Implement a specific SQL Remote feature

This syntax is for modifying a row in one table that affects the partitioning of data in remote databases.

The UPDATE `table-name` syntax makes an entry in the transaction log, but does not change the database table.

The UPDATE `table-name` syntax with no OLD and NEW SUBSCRIBE BY expressions must be used in a BEFORE trigger.

The UPDATE `table-name` syntax with OLD and NEW SUBSCRIBE BY expressions can be used anywhere.

If no OLD and NEW expressions are used, it must be used inside a BEFORE trigger so that it has access to the relevant values. The purpose is to provide a full list of subscribe by values any time the list changes. It is placed in SQL Remote triggers so that the database server can compute the current list of SUBSCRIBE BY values. Both lists are placed in the transaction log.

The Message Agent uses the two lists to make sure that the row moves to any remote database that did not have the row and now needs it. The Message Agent also removes the row from any remote database that has the row and no longer needs it. A remote database that has the row and still needs it is not affected by the UPDATE statement.

The UPDATE `table-name` syntax allows the old SUBSCRIBE BY list and the new SUBSCRIBE BY list to be explicitly specified, which can make SQL Remote triggers more efficient. In the absence of these lists, the database server computes the old SUBSCRIBE BY list from the publication definition. Since the new SUBSCRIBE BY list is commonly only slightly different from the old SUBSCRIBE BY list, the work to compute the old list may be done twice. By specifying both the old and new lists, this extra work can be avoided.

The OLD and NEW SUBSCRIBE BY syntax is especially useful when many tables are being updated in the same trigger with the same subscribe by expressions. This can dramatically increase performance.

The SUBSCRIBE BY expression is either a value or a subquery.

For publications created using a subquery in a subscription expression, you must write a trigger containing the UPDATE `table-name` syntax to ensure that the rows are kept in their proper subscriptions.

The UPDATE `table-name` syntax makes an entry in the transaction log, but does not change the database table.

Privileges

You must be the owner of the table being updated, or have UPDATE privilege on the columns being modified, or have the UPDATE ANY TABLE system privilege.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example uses the syntax for the SQL Remote Message Agent to transfer employee Philip Chin (employee 129) from the sales department to the marketing department.

```
UPDATE GROUPO.Employees
SET DepartmentID = 400
VERIFY ( DepartmentID )
VALUES ( 200 )
WHERE EmployeeID = 129;
```

Related Information

Row Partitioning Among Remote Databases
BEFORE UPDATE Triggers
UPDATE Statement [page 1463]

1.4.4.286 VALIDATE Statement

Validates the current database, one or more tables, materialized views, or indexes in the current database.

Syntax

Validate a database

```
VALIDATE { CHECKSUM | DATABASE }
```

Validate a table or materialized view

```
VALIDATE {
TABLE [ owner.]table-name
| MATERIALIZED VIEW [ owner.]materialized-view-name }
[ WITH EXPRESS CHECK ]
[ WITH DATA LOCK | WITH SNAPSHOT ]
```

Validate an index

```
 VALIDATE {
INDEX index-name
| [ INDEX ] FOREIGN KEY role-name
| [ INDEX ] PRIMARY KEY }
[ WITH DATA LOCK | WITH SNAPSHOT ]
ON [ owner.]object-name
```

```
object-name :
table-name
| materialized-view-name
```

Validate a text index

```
VALIDATE TEXT INDEX index-name
ON [ owner.]table-name
```

Parameters

CHECKSUM clause

Use this clause to validate the checksum on each page of a database. The database must have check sums enabled. The CHECKSUM clause ensures that database pages have not been modified on disk.

When a database is created with checksums enabled, a checksum is calculated for each database page before it is written to disk. CHECKSUM reads each database page directly from disk (not from the database server's cache) and calculates the checksum for each page.

If the calculated checksum for a page does not match the stored checksum for that page, then an error occurs and information about the invalid page appears in the database server messages window.

DATABASE clause

The VALIDATE DATABASE statement ensures that the free map correctly identifies pages as either allocated or free and that no BLOBs are orphaned. VALIDATE DATABASE also performs checksum validation and verifies that each database page belongs to the correct object. For example, on a table page,

the table ID must identify a valid table whose definition must include the current page in its set of table pages.

The VALIDATE DATABASE statement brings pages into the database server's cache in sequential order. The database server always verifies the contents and checksums of pages brought into the cache. If you start database validation while the database cleaner is running, then the validation does not run until the database cleaner is finished running.

[INDEX] PRIMARY KEY | FOREIGN KEY

The VALIDATE INDEX statement performs the same operations as the VALIDATE TABLE statement except that it only validates the specified index and its underlying table; other indexes are not checked.

For foreign key indexes, unless the WITH EXPRESS CHECK clause is specified, each value is looked up in the primary key table to verify that referential integrity is intact. If the specified index is not a foreign key index, then WITH EXPRESS CHECK has no effect.

MATERIALIZED VIEW

Validates the specified materialized view.

TABLE

The VALIDATE TABLE statement validates the specified table and all of its indexes by checking that the set of all rows and values in the base table matches the set of rows and values contained in each index.

VALIDATE TABLE also traverses all the table's BLOBs, verifies BLOB allocation maps, and detects orphaned BLOBs. The VALIDATE TABLE statement checks the physical structure of the table's index pages and verifies the order of the index hash values, and the index's uniqueness requirements (if any are specified).

Because the VALIDATE TABLE statement, like VALIDATE DATABASE, uses the database server's cache, the database server also verifies the checksums and basic validity of all pages in use by a table and its indexes.

TEXT INDEX

The VALIDATE TEXT INDEX statement verifies that the positional information for the terms in the index is intact. If the positional information is not intact, then an error is generated and you must rebuild the text index. If the text index is either auto or manual, then you can rebuild the text index by executing the REFRESH TEXT INDEX statement.

If the generated error concerns an immediate text index, then drop the immediate index and create a new one.

WITH EXPRESS CHECK

Specifying the WITH EXPRESS CHECK clause disables referential integrity checking and can therefore significantly improve performance.

WITH DATA LOCK | WITH SNAPSHOT

When validating tables that have active transactions, choose one of the following options to prevent receiving false errors about corrupt tables:

WITH DATA LOCK

Prevents transactions from modifying the table schema or data by applying exclusive data locks on the specified tables. Concurrent transactions can read, but not modify the table data or schema.

When the FOREIGN KEY clause is specified, then exclusive data locks are also applied to the primary key tables.

WITH SNAPSHOT

Ensures that only committed data is checked by applying snapshot isolation. Transactions can read and modify the data. This clause requires that the database have snapshot isolation enabled (with the allow_snapshot_isolation database option). Because this clause uses snapshot isolation, performance is often affected.

Remarks

> ⚠ Caution
>
> If WITH DATA LOCK or WITH SNAPSHOT is not specified, then perform validation while no connections are making changes to the database; otherwise, false errors may be reported indicating some form of database corruption.

Privileges

You must have the VALIDATE ANY OBJECT system privilege.

Side Effects

When you specify the WITH DATA LOCK clause, then exclusive data locks are applied to the specified tables and views.

When the FOREIGN KEY clause is specified along with the WITH DATA LOCK clause, then exclusive data locks are also applied to the primary key tables.

Automatic commit for the WITH DATA LOCK and WITH SNAPSHOT clauses.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example validates the Products table:

```
VALIDATE TABLE GROUPO.Products;
```

Related Information

Database Validation
Corruption Detection Using Checksums
REFRESH TEXT INDEX Statement [page 1325]
DROP TEXT INDEX Statement [page 1136]
CREATE TEXT INDEX Statement [page 1030]
Validation Utility (dbvalid)
sa_validate System Procedure [page 1748]
sa_clean_database System Procedure [page 1537]
CREATE DATABASE Statement [page 821]
CREATE INDEX Statement [page 886]

1.4.4.287 VALIDATE LDAP SERVER Statement

Validates an LDAP server configuration object.

Syntax

```
VALIDATE LDAP SERVER { ldapua-server-name | ldapua-server-attribs }
[ CHECK user-id [ user-dn-string ] ]
```

Parameters

ldapua-server-name

The name of the LDAP server configuration object to validate. For a full description of this clause, see the CREATE LDAP SERVER statement.

ldapua-server-attribs

When validating an LDAP server configuration object using `ldapua-server-attribs`, the specified attributes are validated. The URLs are parsed to identify syntax errors. Validation stops and an error is returned if a syntax error occurs.

For a full description of this clause, see the CREATE LDAP SERVER statement.

CHECK clause

Use this clause to specify a user ID to search for on the LDAP server.

Remarks

When a VALIDATE LDAP SERVER statement is executed, a connection to the LDAP server is attempted. If ACCESS ACCOUNT and a password are specified, the values are used to establish the connection to the SEARCH DN URL, validating the SEARCH DN URL, ACCESS ACCOUNT, and ACCESS ACCOUNT password.

When setting up a new server to use LDAP User Authentication, this statement is useful validating changes to an LDAP server configuration object before applying them, and for diagnosing problems between the database server and the LDAP server.

If you use this statement in a procedure and include an IDENTIFIED BY clause in the attributes, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side Effects

Automatic commit.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example creates and LDAP server configuration object and connects to the LDAP server at hostname voyager, port number 389, using the ACCESS ACCOUNT and password specified in the definition for apps_primary.

```
CREATE LDAP SERVER apps_primary2
    SEARCH DN
        URL  'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*'
        ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
        IDENTIFIED BY 'Secret99Password'
    AUTHENTICATION URL 'ldap://voyager:389/'
    CONNECTION TIMEOUT 3000
    WITH ACTIVATE;
```

```
VALIDATE LDAP SERVER apps_primary2;
```

The following example connects to the LDAP server at hostname voyager, port number 389, using the ACCESS ACCOUNT and password specified in the definition for apps_primary2. It also checks that user ID myusername is valid and matches the expected user DN:

```
VALIDATE LDAP SERVER apps_primary2
CHECK myusername 'cn=myusername,cn=Users,dc=mycompany,dc=com';
```

If the LDAP server configuration object has not been defined, the same checks can be performed by specifying the attributes:

```
VALIDATE LDAP SERVER
SEARCH DN
URL 'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*'
ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
IDENTIFIED BY 'Secret99Password'
AUTHENTICATION URL 'ldap://voyager:389/'
CONNECTION TIMEOUT 3000
CHECK myusername 'cn=myusername,cn=Users,dc=mycompany,dc=com';
```

Related Information

CREATE LDAP SERVER Statement [page 891]
ALTER LDAP SERVER Statement [page 687]
DROP LDAP SERVER Statement [page 1100]

1.4.4.288 WAITFOR Statement

Delays processing for the current connection for a specified amount of time or until a given time.

Syntax

```
WAITFOR {
DELAY time
| TIME time }
[ CHECK EVERY integer ]
[ AFTER MESSAGE BREAK ]
```

```
time : string
```

Parameters

DELAY clause

If DELAY is used, processing is suspended for the given interval.

TIME clause

If TIME is specified, processing is suspended until the database server time reaches the time specified. If the current server time is greater than the time specified, processing is suspended until that time on the following day.

CHECK EVERY clause

This optional clause controls how often the WAITFOR statement wakes up. By default, it wakes up every 5 seconds. The value is in milliseconds, and the minimum value is 250 milliseconds.

AFTER MESSAGE BREAK clause

The WAITFOR statement can be used to wait for a message from another connection. When a message is received it is usually forwarded to the application that executed the WAITFOR statement and the WAITFOR statement continues to wait. If the AFTER MESSAGE BREAK clause is specified, when a message is received from another connection, the WAITFOR statement completes. The message text is not forwarded to the application, but it can be accessed by obtaining the value of the MessageReceived connection property.

Remarks

The WAITFOR statement wakes up periodically (every 5 seconds by default) to check if it has been canceled or if messages have been received. If neither of these has happened, the statement continues to wait.

Because scheduled events execute on their own connection, scheduled events are often a better choice than using WAITFOR TIME.

Privileges

None

Side Effects

The implementation of the WAITFOR statement causes the worker servicing the statement to block while it is waiting. This reduces the number of available workers in the worker pool.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following example waits for three seconds:

```
WAITFOR DELAY '00:00:03';
```

The following example waits for 0.5 seconds (500 milliseconds):

```
WAITFOR DELAY '00:00:00.500';
```

The following example waits until 8 PM:

```
WAITFOR TIME '20:00';
```

In the following example, connection 1's WAITFOR statement completes when it receives the message from connection 2:

```
// connection 1:
BEGIN
  DECLARE msg LONG VARCHAR;
  LOOP  // forever
    WAITFOR DELAY '00:05:00' AFTER MESSAGE BREAK;
    SET msg = CONNECTION_PROPERTY('MessageReceived');
    IF msg != '' THEN
      MESSAGE 'Msg: ' || msg TO CONSOLE;
    END IF;
  END LOOP
END;
// connection 2:
MESSAGE 'here it is' FOR connection 1
```

Related Information

Threading
CREATE EVENT Statement [page 847]
MESSAGE Statement [page 1278]

1.4.4.289 WAITFOR SEMAPHORE Statement

Decrements the counter associated with a semaphore.

Syntax

```
WAITFOR SEMAPHORE [owner.]semaphore-name
[ TIMEOUT number-milliseconds ]
```

Parameters

owner

The owner of the semaphore. `owner` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

semaphore-name

The name of the semaphore. `semaphore-name` can also be specified using an indirect identifier (for example, `` `[@variable-name]` ``).

TIMEOUT clause

Specify the duration of time, in milliseconds, to wait to decrement the counter associated with the semaphore. If this clause is not specified, then the connection waits indefinitely until the count can be decremented, or until an error is returned.

`number-milliseconds` can be specified using a variable (for example, **`TIMEOUT @timeout-value`**). If `number-milliseconds` is set to a variable and the variable is NULL, the behavior is equivalent to not specifying the clause.

Remarks

The WAITFOR SEMAPHORE statement decrements the counter associated with the semaphore if it is not currently zero; otherwise, it blocks.

If the counter is a positive integer, then the counter is decremented by 1 and the statement completes.

If the counter is 0, then the connection waits until notified that the counter is a positive integer, or until the duration specified by the TIMEOUT clause passes, at which point an error is returned indicating the timeout. Connections are notified in first-in, first-out (FIFO) order of WAITFOR statement execution.

An error is returned if the current connection is identified during deadlock detection while waiting on the semaphore. An error is also returned if the semaphore is dropped.

If a connection that notified a semaphore is dropped or canceled, the counter increment persists.

Privileges

You must have the UPDATE ANY MUTEX SEMAPHORE system privilege or be the owner of the semaphore.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement creates a semaphore and sets its initial value to zero:

```
CREATE OR REPLACE SEMAPHORE DBA.gate START WITH 0;
```

The following statements define a stored procedure that waits on this semaphore. If the semaphore counter is not zero, it will decrement the counter and proceed. If the semaphore counter is zero, it wil block:

```
CREATE OR REPLACE PROCEDURE SemTest()
BEGIN
    WAITFOR SEMAPHORE DBA.gate;
    SELECT 'Waitfor done';
END;
```

The following statement calls the stored procedure. Execution will be delayed since the semaphore is initially zero and the stored procedure blocks. It will block until a NOTIFY SEMAPHORE statement is executed on another connection.

```
CALL SemTest;
```

Execute the following statement on a separate connection. It increments the semaphore by 1 and allows a blocked connection to proceed. If no connection is blocked, the next WAITFOR statement that references the semaphore will proceed without blocking.

```
NOTIFY SEMAPHORE DBA.gate INCREMENT BY 1;
```

Execute the following statement on a separate connection. It increments the semaphore by 2 and allows two blocked connections to proceed. If only one connection is blocked, the next WAITFOR statement that references the semaphore will proceed without blocking. If no connections are blocked, the next 2 WAITFOR statements that reference the semaphore will proceed without blocking.

```
NOTIFY SEMAPHORE DBA.gate INCREMENT BY 2;
```

Related Information

Mutexes and Semaphores
DROP SEMAPHORE Statement [page 1118]
NOTIFY SEMAPHORE Statement [page 1282]
CREATE SEMAPHORE Statement [page 957]
SYSMUTEXSEMAPHORE System View [page 1925]

1.4.4.290 WHENEVER Statement [ESQL]

Specifies error handling in Embedded SQL programs.

Syntax

```
WHENEVER { SQLERROR | SQLWARNING | NOTFOUND }
GOTO
    label
    | STOP
    | CONTINUE
    | { C-code; }
```

```
label : identifier
```

Remarks

The WHENEVER statement is used to trap errors, warnings and exceptional conditions encountered by the database when processing SQL statements. The statement can be put anywhere in an Embedded SQL program and does not generate any code. The preprocessor will generate code following each successive SQL statement. The error action remains in effect for all Embedded SQL statements from the source line of the WHENEVER statement until the next WHENEVER statement with the same error condition, or the end of the source file.

Note

The error conditions are in effect based on positioning in the C language source file, not based on when the statements are executed.

The default action is CONTINUE.

This statement is provided for convenience in simple programs. Most of the time, checking the sqlcode field of the SQLCA (SQLCODE) directly is the easiest way to check error conditions. In this case, the WHENEVER statement would not be used. In fact, all the WHENEVER statement does is cause the preprocessor to generate an `if ( SQLCODE )` test after each statement.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

An exception condition declaration made with the WHENEVER statement is a Core Feature. The standard uses the keyword SQLEXCEPTION rather than SQLERROR. The ability to directly include C code in the WHENEVER statement, rather than merely a statement label, is not in the standard. The action STOP is also not in the standard.

Example

```
EXEC SQL WHENEVER NOTFOUND GOTO done;
EXEC SQL WHENEVER SQLERROR
   {
      PrintError( &sqlca );
      return( FALSE );
   };
```

1.4.4.291 WHILE Statement [T-SQL]

Provides repeated execution of a statement or compound statement.

Syntax

```
WHILE search-condition statement
```

Remarks

The WHILE conditional affects the execution of only a single SQL statement, unless statements are grouped into a compound statement between the keywords BEGIN and END.

The BREAK statement and CONTINUE statement can be used to control execution of the statements in the compound statement. The BREAK statement terminates the loop, and execution resumes after the END keyword marking the end of the loop. The CONTINUE statement causes the WHILE loop to restart, skipping any statements after the CONTINUE.

Privileges

None.

Side Effects

None.

Standards

ANSI/ISO SQL Standard

The WHILE statement is part of optional ANSI/ISO SQL language feature P002, "Computational completeness". The Transact-SQL variant of the WHILE statement does not include END WHILE.

Example

The following code illustrates the use of WHILE:

```
WHILE ( SELECT AVG(UnitPrice) FROM GROUPO.Products ) < $30
BEGIN
   UPDATE GROUPO.Products
   SET UnitPrice = UnitPrice + 2
   IF ( SELECT MAX(UnitPrice) FROM GROUPO.Products ) > $50
      BREAK
END
```

The BREAK statement breaks the WHILE loop if the most expensive product has a price above $50. Otherwise, the loop continues until the average price is greater than or equal to $30.

Related Information

LOOP Statement [page 1269]
CONTINUE Statement [page 818]

1.4.4.292 WINDOW Clause

Defines all or part of a window for use with window functions such as AVG and RANK in a SELECT statement.

Syntax

```
WINDOW window-expression, ...

window-expression : new-window-name AS ( window-spec )

window-spec :
[ existing-window-name ]
[ PARTITION BY expression, ... ]
[ ORDER BY expression [ ASC | DESC ], ... ]
[ { ROWS | RANGE } { window-frame-start | window-frame-between } ]

window-frame-start :
{ UNBOUNDED PRECEDING
  | unsigned-integer PRECEDING
  | CURRENT ROW }

window-frame-between :
BETWEEN window-frame-bound1 AND window-frame-bound2

window-frame-bound :
window-frame-start
| UNBOUNDED FOLLOWING
| unsigned-integer FOLLOWING
```

Parameters

PARTITION BY clause

The PARTITION BY clause organizes the result set into logical groups based on the unique values of the specified expression. When this clause is used with window functions, the functions are applied to each partition independently. For example, if you follow PARTITION BY with a column name, the result set is partitioned by distinct values in the column.

If this clause is omitted, the entire result set is considered a partition.

The PARTITION BY `expression` cannot be an integer literal.

ORDER BY clause

The ORDER BY clause defines how to sort the rows in each partition of the result set. You can further control the order by specifying ASC for ascending order (the default) or DESC for descending order.

The ORDER BY `expression` cannot be an integer literal.

If this clause is omitted, the database server returns rows in whatever order is most efficient, and the appearance of result sets may vary depending on when you last accessed the row.

ROWS clause and RANGE clause

Use either a ROWS or RANGE clause to express the size of the window. The window size can be one, many, or all rows of a partition. You can express the size of the window as a range of data values offset from the value in the current row (RANGE), or the number of physical rows offset from the current row (ROWS).

When using the RANGE clause, you must also specify an ORDER BY clause because range calculations require values to be sorted. The ORDER BY clause for ranges must contain one expression, and that expression must result in either a date or a numeric value.

If you do not specify a ROWS or RANGE clause, the database server uses default window sizes based on whether an ORDER BY clause is present.

PRECEDING clause

Use the PRECEDING clause to define the first row of the window using the current row as a reference point. The starting row is expressed as the number of rows preceding the current row. For example, `5 PRECEDING` sets the window to start with the fifth row preceding the current row.

Use UNBOUNDED PRECEDING to set the first row in the window to be the first row in the partition.

BETWEEN clause

Use the BETWEEN clause to define the first and last row of the window, using the current row as a reference point. First and last rows are expressed as the number of rows preceding and following the current row, respectively. For example, `BETWEEN 3 PRECEDING AND 5 FOLLOWING` sets the window to start with the third row preceding the current row, and end with the fifth row following the current row.

Use BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING to set the first and last rows in the window to be the first and last row in the partition, respectively. This is equivalent to the default behavior if no ROW or RANGE clause is specified.

FOLLOWING clause

Use the FOLLOWING clause to define the last row of the window using the current row as a reference point. The last row is expressed as the number of rows following the current row.

Use UNBOUNDED FOLLOWING to set the last row in the window to be the last row in the partition.

Remarks

The WINDOW clause must appear before the ORDER BY clause in a SELECT statement.

With the exception of the LIST function, all aggregate functions can be used as window functions. However, ranking aggregate functions (RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST, and ROW_NUMBER) require an ORDER BY clause, and do not allow a ROW or RANGE clause in the WINDOW clause or inline definition. For all other window functions, you can use any of the clauses.

Privileges

None.

Standards

ANSI/ISO SQL Standard

The WINDOW clause and window aggregate functions comprise optional SQL Language Features T611, "Elementary OLAP operations", and T612, "Advanced OLAP operations". The window functions FIRST_VALUE and LAST_VALUE are not in the standard.

Example

The following example returns an employee's salary and the average salary for all employees in the selected state. The results are ordered by state and then by surname.

```
SELECT EmployeeID, Surname, Salary, State,
  AVG( Salary ) OVER Salary_Window
FROM GROUPO.Employees
WINDOW Salary_Window AS ( PARTITION BY State )
ORDER BY State, Surname;
```

Related Information

OLAP Support
Window Definitions
Window Definition: Inlining Using the OVER Clause and WINDOW Clause
SELECT Statement [page 1362]

1.4.4.293 WRITETEXT Statement [T-SQL]

Permits non-logged updating of a CHAR, NCHAR, or BINARY column. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

Syntax

```
WRITETEXT table-name.column-name
text-pointer [ WITH LOG ] data
```

Remarks

Updates an existing column value. The update is not recorded in the transaction log, unless the WITH LOG option is supplied. You cannot perform WRITETEXT operations on views.

Privileges

None.

Side Effects

WRITETEXT does not fire triggers, and by default WRITETEXT operations are not recorded in the transaction log.

Standards

ANSI/ISO SQL Standard

Transact-SQL extension.

Example

The following Embedded SQL code fragment illustrates the use of the WRITETEXT statement. The SELECT statement in this example returns a single row. The example replaces the contents of the Description column on the specified row with a new value.

```
EXEC SQL create variable txtptr binary(16);
EXEC SQL set txtptr =
    ( SELECT textptr(Description)
        FROM MarketingInformation
        WHERE ProductID = '500' );
EXEC SQL writetext MarketingInformation.Description
   txtptr 'newdata';
```

Related Information

READTEXT Statement [T-SQL] [page 1320]
TEXTPTR Function [Text and Image] [page 588]

1.5 Tables

There are several types of tables supported by the software.

In this section:

1.5.1 System Tables

The structure of every database is described in system tables.

System tables are owned by the user SYS. The contents of these tables can be changed only by the database server. The UPDATE, DELETE, and INSERT statements cannot be used to modify the contents of these tables. Further, the structure of these tables cannot be changed using the ALTER TABLE and DROP statements. System views are updated when a checkpoint occurs.

With a few exceptions, data in system tables are exposed via their corresponding **system views**. Each system table name start with capital i (I). Each corresponding system view has the same name but without the i at the beginning. To view the information about a system table, search the documentation for the corresponding system view.

In this section:

Related Information

System Views [page 1889]
MobiLink Server System Tables
SQL Remote System Tables

1.5.1.1 Exceptional Tables You Can Access

There are several system tables that allow you to access them directly.

In this section:

1.5.1.1.1 DUMMY System Table

The DUMMY table is provided as a read-only table that always has exactly one row.

Column name	Column type	Column constraint	Table constraints
dummy_col	INTEGER	NOT NULL	

This can be useful for extracting information from the database, as in the following example that gets the current user ID and the current date from the database.

```
SELECT USER, today(*) FROM SYS.DUMMY;
```

dummy_col

This column is not used. It is present because a table cannot be created with no columns.

The cost of reading from the DUMMY table is less than the cost of reading from a similar user-created table because there is no lock placed on the table page of DUMMY.

Access plans are not constructed with scans of the DUMMY table. Instead, references to DUMMY are replaced with a Row Constructor algorithm, which virtualizes the table reference. This eliminates contention associated with the use of DUMMY. DUMMY still appears as the table and/or correlation name in short, long, and graphical plans.

1.5.1.1.2 Diagnostic Tracing Tables (Deprecated)

Following are the main tables that are used for diagnostic tracing. These tables are owned by the dbo user, and are not considered system tables.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

Unlike system tables, diagnostic tables do not have corresponding views for viewing their data. You can query the tables directly.

For most of these tables, there exists a global shared temporary table with a similar name and schema. For example, the sa_diagnostic_blocking table has a global temporary table counterpart, sa_tmp_diagnostic_blocking table, which has the same schema. During a tracing session, diagnostic data is written to these temporary tables. Because temporary tables are not logged, they provide superior performance during a tracing session, where it is important to minimize the impact on the server.

In this section:

sa_diagnostic_auxiliary_catalog Table (Deprecated) [page 1496]
The sa_diagnostic_auxiliary_catalog table is owned by the dbo user, and is used to map database objects between the production database and tracing database. Objects include user tables, procedures, and functions. This table is used primarily by the Index Consultant and the TRACED_PLAN function.

sa_diagnostic_blocking Table (Deprecated) [page 1496]
The sa_diagnostic_blocking table is owned by the dbo user, and records blocking events.

sa_diagnostic_cachecontents Table (Deprecated) [page 1497]
The sa_diagnostic_cachecontents table is owned by the dbo user.

sa_diagnostic_connection Table (Deprecated) [page 1498]
The sa_diagnostic_connection table is owned by the dbo user, and has one row for every database connection that is active during the logging session. Connect and disconnect times, if they occur within the logging session, can be derived from the sa_diagnostic_request table.

sa_diagnostic_cursor Table (Deprecated) [page 1499]
The sa_diagnostic_cursor table is owned by the dbo user. Each row describes either an internal or external cursor opened during the logging session.

sa_diagnostic_deadlock Table (Deprecated) [page 1500]
The sa_diagnostic_deadlock table is owned by the dbo user.

sa_diagnostic_hostvariable Table (Deprecated) [page 1501]
The sa_diagnostic_hostvariable table is owned by the dbo user, and contains the values of host variables used by the specified cursor.

sa_diagnostic_internalvariable Table (Deprecated) [page 1502]
The sa_diagnostic_internalvariable table is owned by the dbo user, and contains the values of internal (local) variables used by a given statement. This table is primarily used by the Index Consultant, and the traced_plan function.

sa_diagnostic_query Table (Deprecated) [page 1503]
The sa_diagnostic_query table is owned by the dbo user, and stores optimization information for queries, especially the context in which they were optimized. A row in this table represents an invocation of the optimizer for a query. Plans captured at optimization time are stored here.

sa_diagnostic_request Table (Deprecated) [page 1505]
The sa_diagnostic_request table is owned by the dbo user, and is the master table for all requests.

sa_diagnostic_statement Table (Deprecated) [page 1506]
The sa_diagnostic_statement table is owned by the dbo user, and stores the text of statements.

sa_diagnostic_statistics Table (Deprecated) [page 1507]
The sa_diagnostic_statistics table is owned by the dbo user, and contains a history of performance counters maintained in the server. Each row represents the value of a given performance counter at a given moment in time.

sa_diagnostic_tracing_level Table (Deprecated) [page 1508]
The sa_diagnostic_tracing_level table is owned by the dbo user, and each row in this table is a condition that determines what kind of diagnostic information to send to the tracing database.

Related Information

SQL Anywhere Profiler
Diagnostic Tracing (Deprecated)
SQL Anywhere Profiler

1.5.1.1.2.1 sa_diagnostic_auxiliary_catalog Table (Deprecated)

The sa_diagnostic_auxiliary_catalog table is owned by the dbo user, and is used to map database objects between the production database and tracing database. Objects include user tables, procedures, and functions. This table is used primarily by the Index Consultant and the TRACED_PLAN function.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

Columns

Column name	Column type	Description
original_object_id	UNSIGNED BIGINT	The object ID of this object in the main tracing database.
local_object_id	UNSIGNED BIGINT	The object ID of this object in the auxiliary tracing database.
pages_if_table	UNSIGNED INT	If the object is a table, this is the number of pages in the table. If the object is not a table, this value is NULL.
rows_if_table	UNSIGNED BIGINT	If the object is a table, this is the number of rows in the table. If the object is not a table, this value is NULL.

Related Information

Optimize Indexes to Improve Performance
SQL Anywhere Profiler
TRACED_PLAN Function [Miscellaneous] (Deprecated) [page 598]

1.5.1.1.2.2 sa_diagnostic_blocking Table (Deprecated)

The sa_diagnostic_blocking table is owned by the dbo user, and records blocking events.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

If logging of blocking events is enabled, a row is inserted in this table each time a connection is blocked while trying to access a resource. Typically, this is caused by either a table or a row lock. A large number of blocks may indicate that you must examine the concurrency in your application to reduce contention for tables and rows.

There are two versions of this table: sa_diagnostic_blocking, and sa_tmp_diagnostic_blocking.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
lock_id	UNSIGNED BIGINT	The ID of the lock that caused the blocking if a row or table lock caused the block, otherwise NULL.
request_id	UNSIGNED BIGINT	The ID of the request that was blocked if the block did not occur because of a cursor, otherwise NULL. This value corresponds to the ID assigned to the request in sa_diagnostic_request.
cursor_id	UNSIGNED BIGINT	The ID of the cursor if the block occurred because of a cursor, otherwise NULL. This value corresponds to the ID assigned to the cursor in sa_diagnostic_cursor.
original_table_object_id	UNSIGNED BIGINT	If the block occurred because of a table lock, the ID of the table on which the block occurred, otherwise NULL.
rowid	UNSIGNED BIGINT	If the block occurred because of a row lock, the ID of the row on which the block occurred, otherwise NULL.
block_time	TIMESTAMP	The time at which the block occurred.
unblock_time	TIMESTAMP	The time at which the block ended.
blocked_by	UNSIGNED INT	The ID of the connection that held the lock, causing the block.

Related Information

Transaction Blocking and Deadlock
How Locking Works
SQL Anywhere Profiler

1.5.1.1.2.3 sa_diagnostic_cachecontents Table (Deprecated)

The sa_diagnostic_cachecontents table is owned by the dbo user.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

When diagnostic tracing is enabled, periodic snapshots of the cache contents are taken. The sa_diagnostic_cachecontents table records the number of table pages for each table in the cache at the time

the snapshot was taken, and the number of rows in each table. The optimizer can use this information to recreate the conditions under which a query was originally optimized, and then make optimization decisions.

Data in the sa_diagnostic_cachecontents table is updated every 20 seconds, as long as there is query activity.

There are two versions of this table: sa_diagnostic_cachecontents, and sa_tmp_diagnostic_cachecontents.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
"time"	TIMESTAMP	The time at which the snapshot of the cache was taken.
original_table_object_id	UNSIGNED BIGINT	The object ID of each table represented in the snapshot.
pages_in_cache	UNSIGNED INT	For a specified table in the snapshot, the total number of pages in cache at the moment of the snapshot.
num_table_pages	UNSIGNED INT	For a specified table in the snapshot, the total number of pages for the table.
num_table_rows	UNSIGNED BIGINT	For a specified table in the snapshot, the total number of rows in the table.

1.5.1.1.2.4 sa_diagnostic_connection Table (Deprecated)

The sa_diagnostic_connection table is owned by the dbo user, and has one row for every database connection that is active during the logging session. Connect and disconnect times, if they occur within the logging session, can be derived from the sa_diagnostic_request table.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

Most of the values in this table mirror values of connection properties.

There are two versions of this table: sa_diagnostic_connection, and sa_tmp_diagnostic_connection.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
connection_number	UNSIGNED INT	A number assigned by the database server to identify the connection of a specific user to the database. This value reflects the value of the Number connection property.
connection_name	LONG VARCHAR	Optional name property for the connection. This value reflects the value of the Name connection property.
user_name	LONG VARCHAR	The name of the user connected to the database.
comm_link	CHAR(40)	Specifies the client-side network protocol options. This value reflects the value of the CommLinks connection property.
node_address	LONG VARCHAR	The node for the client in a client/server connection. This value reflects the value of the NodeAddress connection property.
appinfo	LONG VARCHAR	Information about the client process, such as the IP address of the client computer, the operating system it is running on, and so on. This value reflects the value of the AppInfo connection property.

Related Information

SQL Anywhere Profiler
List of Connection Properties

1.5.1.1.2.5 sa_diagnostic_cursor Table (Deprecated)

The sa_diagnostic_cursor table is owned by the dbo user. Each row describes either an internal or external cursor opened during the logging session.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

There are two versions of this table: sa_diagnostic_cursor, and sa_tmp_diagnostic_cursor.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
cursor_id	UNSIGNED BIGINT	A unique number identifying the cursor.
query_id	UNSIGNED BIGINT	Identifies the query over which this cursor ranges.
isolation_level	TINYINT	Isolation level at which this cursor was opened.
flags	UNSIGNED INT	Internal use.
forward_fetches	UNSIGNED INT	Number of forward fetches, including prefetches, done on the cursor.
reverse_fetches	UNSIGNED INT	Number of reverse fetches, including prefetches, done on the cursor.
absolute_fetches	UNSIGNED INT	Number of absolute fetches done on the cursor.
first_fetch_time_ms	UNSIGNED INT	Duration of time spent fetching the first row.
total_fetch_time_ms	UNSIGNED INT	Duration of time spent fetching. This value does not include application processing time between actual fetches (think time).
plan_xml	LONG VARCHAR	Detailed plan for cursors that were dumped at the time the cursor was closed. These plans contain detailed statistics where appropriate.

Related Information

Cursor Usage
SQL Anywhere Profiler

1.5.1.1.2.6 sa_diagnostic_deadlock Table (Deprecated)

The sa_diagnostic_deadlock table is owned by the dbo user.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

When diagnostic tracing is enabled and is set to include tracing of deadlock events, a set of rows is inserted into this table every time a deadlock occurs (one row for each connection that was part of the deadlock is inserted). The set of all rows that comprise a single deadlock event is uniquely identified by a snapshot_id.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
snapshot_id	UNSIGNED BIGINT	A number identifying which deadlock event this row is a part of. This column has nothing to do with snapshot isolation.
snapshot_at	TIMESTAMP	The time at which the deadlock occurred.
waiter	UNSIGNED INT	The connection number of the connection that this row represents.
request_id	UNSIGNED BIGINT	The ID of the request that this connection was processing when the deadlock occurred.
original_table_object_id	UNSIGNED BIGINT	The object ID of the table on which this connection was blocked.
rowid	UNSIGNED BIGINT	The record ID of the row on which this connection was blocked.
owner	UNSIGNED INT	The connection number of the connection that locked the desired row.
rollback_operation_count	UNSIGNED INT	The number of uncommitted operations.

Related Information

Transaction Blocking and Deadlock
SQL Anywhere Profiler

1.5.1.1.2.7 sa_diagnostic_hostvariable Table (Deprecated)

The sa_diagnostic_hostvariable table is owned by the dbo user, and contains the values of host variables used by the specified cursor.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

There are two versions of this table: sa_diagnostic_hostvariable, and sa_tmp_diagnostic_hostvariable.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
request_id	UNSIGNED BIGINT	The ID of the request to which the host variables belong.
cursor_id	UNSIGNED BIGINT	The ID of the cursor to which the host variables pertain.
hostvar_num	UNSIGNED SMALLINT	The ordinal position of the host variable in the SQL statement.
hostvar_type	UNSIGNED TINYINT	For internal use only.
hostvar_value	LONG NVARCHAR	A string representing the value of the host variable. Even if the host variable is an integer or a float, the value is still represented here as a string.

Related Information

Host Variables in Embedded SQL
SQL Anywhere Profiler

1.5.1.1.2.8 sa_diagnostic_internalvariable Table (Deprecated)

The sa_diagnostic_internalvariable table is owned by the dbo user, and contains the values of internal (local) variables used by a given statement. This table is primarily used by the Index Consultant, and the traced_plan function.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

There are two versions of this table: sa_diagnostic_internalvariable, and sa_tmp_diagnostic_internalvariable.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
request_id	UNSIGNED BIGINT	The ID of the request that contains the internal variable.
rowvariable_id	UNSIGNED INT	The column number in the row variable of this value.
variable_domain	UNSIGNED SMALLINT	For internal use only.
variable_name	CHAR(128)	The name of the internal variable.
variable_value	LONG NVARCHAR	A string representing the value of the internal variable.

Related Information

SQL Variables [page 123]
SQL Anywhere Profiler

1.5.1.1.2.9 sa_diagnostic_query Table (Deprecated)

The sa_diagnostic_query table is owned by the dbo user, and stores optimization information for queries, especially the context in which they were optimized. A row in this table represents an invocation of the optimizer for a query. Plans captured at optimization time are stored here.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

Some of the values in this table mirror database option values.

There are two versions of this table: sa_diagnostic_query, and sa_tmp_diagnostic_query.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	The ID of the logging session during which the query or request occurred.

Column name	Column type	Description
query_id	UNSIGNED BIGINT	A number uniquely identifying the query.
statement_id	UNSIGNED BIGINT	A number uniquely identifying a statement in a query.
user_object_id	UNSIGNED BIGINT	The object ID of the user under which this query was executed. If the query was run from a procedure, this would be the user ID of the procedure owner.
start_time	TIMESTAMP	The time at which this query was optimized.
cache_size_bytes	UNSIGNED BIGINT	The size, in bytes, of the cache at the time this query was optimized.
optimization_goal	TINYINT	Determines whether query processing is optimized towards returning the first row quickly, or minimizing the cost of returning the complete result set. This value reflects the value of the optimization_goal database option.
optimization_level	TINYINT	Controls the amount of effort made by the SQL Anywhere query optimizer to find an access plan for a SQL statement. This value reflects the value of the optimization_level database option.
user_estimates	TINYINT	Controls whether user selectivity estimates in query predicates are respected or ignored by the query optimizer. This value reflects the value of the user_estimates database option.
optimization_workload	TINYINT	Determines whether query processing is optimized towards a workload that is a mix of updates and reads or a workload that is predominantly read-based. This value reflects the value of the optimization_workload database option.
available_requests	TINYINT	Used internally to compute the level of intra-query parallelism.
active_requests	TINYINT	Used internally to compute the level of intra-query parallelism.
max_tasks	TINYINT	Used internally to compute the level of intra-query parallelism.

Column name	Column type	Description
used_bypass	TINYINT	Whether a simple query bypass was used. A value of 1 indicates a bypass was used; a value of 0 indicates that the query was fully optimized.
estimated_cost_ms	TINYINT	The estimated cost, in milliseconds.
plan_explain	LONG VARCHAR	A text plan representation of this query.
plan_xml	LONG VARCHAR	A graphical plan representation of the query (if one was recorded).
sql_rewritten	LONG VARCHAR	Text of a query after applying optimizations. A value will only be present in this column if optimization logging is enabled.

Related Information

Database Options
How the Optimizer Works
SQL Anywhere Profiler
optimization_level Option
optimization_workload Option
user_estimates Option

1.5.1.1.2.10 sa_diagnostic_request Table (Deprecated)

The sa_diagnostic_request table is owned by the dbo user, and is the master table for all requests.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

A request is an event related to query processing and generally includes

:

- connect or disconnect events
- statement executions
- statement preparations
- open or drop cursor events

There are two versions of this table: sa_diagnostic_request and sa_tmp_diagnostic_request.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	The logging session during which the request occurred.
request_id	UNSIGNED BIGINT	A number uniquely identifying the request.
start_time	TIMESTAMP	The time at which the event started.
finish_time	TIMESTAMP	For statement execution, the time when the statement completed; otherwise, NULL.
duration_ms	UNSIGNED INT	The duration of the event in milliseconds.
connection_number	UNSIGNED INT	The ID of the connection that caused the event to happen.
request_type	UNSIGNED SMALLINT	The type of request. Values include: New diagnostic tracing session started, SQL Statement executed, Cursor opened, Cursor closed, Client connect, Client disconnect, and Checkpoint.
statement_id	UNSIGNED BIGINT	If the event was statement-related, the ID assigned to the statement for tracing purposes.
query_id	UNSIGNED BIGINT	If the event was query-related, the ID assigned to the query for tracing purposes.
cursor_id	UNSIGNED BIGINT	If the event was cursor-related, the ID assigned to the cursor for tracing purposes.
sql_code	SMALLINT	Since rows in this table represent operations on statements, cursors, or queries, most return a SQL code. This column contains the SQL code returned. If a SQL code of 0 is returned, the column contains NULL.

1.5.1.1.2.11 sa_diagnostic_statement Table (Deprecated)

The sa_diagnostic_statement table is owned by the dbo user, and stores the text of statements.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

A row in this table represents a SQL statement that was executed by the server. Such statements may have been issued by an external source, such as a client request, or by an internal source such as a procedure, trigger, or user-defined function. Internal statements only appear here once per session.

There are two versions of this table: sa_diagnostic_statement, and sa_tmp_diagnostic_statement.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	The logging session during which the statement was submitted.
statement_id	UNSIGNED BIGINT	A unique number assigned to the statement for tracing purposes.
database_object	UNSIGNED BIGINT	If the statement came from a procedure, trigger, or function, this is the ID as specified in the ISYSOBJECT system table.
line_number	UNSIGNED SMALLINT	If the statement formed part of a compound statement, this reflects the ordinal position of the statement within the compound statement.
signature	UNSIGNED INT	Used internally to group similar queries.
statement_text	LONG VARCHAR	The statement text.

1.5.1.1.2.12 sa_diagnostic_statistics Table (Deprecated)

The sa_diagnostic_statistics table is owned by the dbo user, and contains a history of performance counters maintained in the server. Each row represents the value of a given performance counter at a given moment in time.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

There are two versions of this table: sa_diagnostic_statistics, and sa_tmp_diagnostic_statistics.

Columns

Column name	Column type	Description
logging_session_id	UNSIGNED INT	A number uniquely identifying the logging session during which the diagnostic information was gathered.
"time"	TIMESTAMP	The time at which the performance counter value was captured.

Column name	Column type	Description
counter_id	UNSIGNED SMALLINT	A number uniquely identifying the performance counter. You can get the name of the property that this counter_id represents using the PROPERTY_NAME function.
type	TINYINT	Indicates whether this is a database, server, or connection statistic. Possible values are 0 for server, 1 for database, 2 for connection, and 4 for external database.
connection_number	UNSIGNED INT	For a connection statistic, the connection number from which this property was captured. For an extended database statistic, the file number for the file from which this property was captured. Otherwise, the value is 0.
counter_value	UNSIGNED INT	The value of the performance counter.

Related Information

SQL Anywhere Profiler
PROPERTY_NAME Function [System] [page 495]

1.5.1.1.2.13 sa_diagnostic_tracing_level Table (Deprecated)

The sa_diagnostic_tracing_level table is owned by the dbo user, and each row in this table is a condition that determines what kind of diagnostic information to send to the tracing database.

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

If logging data meets the conditions of one or more rows in this table, then the corresponding data is logged.

Data in this table is populated using the CONNECT TRACING or REFRESH TRACING LEVEL statements.

Columns

Column name	Column type	Description
id	UNSIGNED INT	For internal use only.

Column name	Column type	Description
scope	CHAR(32)	The scope of the diagnostic tracing, as listed below. • DATABASE • ORIGIN • USER • CONNECTION_NAME • CONNECTION_NUMBER • FUNCTION • PROCEDURE • EVENT • TRIGGER • TABLE
identifier	CHAR(128)	The identifier for the scope. This value changes, depending on the specified scope. For example: • if `scope` is DATABASE, `identifier` may not be present. • if `scope` is ORIGIN, `identifier` must be either Internal or External. • if `scope` is USER, `identifier` is the ID of the user. • if `scope` is CONNECTION_NAME, or CONNECTION_NUMBER, `identifier` is the name or number, respectively, for the connection. • if `scope` is FUNCTION, PROCEDURE, EVENT, TRIGGER, or TABLE, `identifier` is the fully qualified identifier for the object.
trace_type	CHAR(32)	The type of data to trace for the specified scope, as listed below. • VOLATILE_STATISTICS • NONVOLATILE_STATISTICS • CONNECTION_STATISTICS • BLOCKING • PLANS • PLANS_WITH_STATISTICS • STATEMENTS • STATEMENTS_WITH_VARIABLES

Column name	Column type	Description
trace_condition	CHAR(32)	Applies only to plans, and controls whether to trace large, expensive queries, or queries for which the optimizer did not make optimal choices. Possible values are listed below. • NONE, or NULL • SAMPLE_EVERY • ABSOLUTE_COST • RELATIVE_COST_DIFFERENCE
value	UNSIGNED INT	The value associated with the `condition`. For example, if `condition` is SAMPLE_EVERY, the `condition_value` would be a positive integer reflecting time in milliseconds. Additional rules are as follows: • If `condition` is NULL or NONE, there is no `condition_value`. • If `condition` is ABSOLUTE_COST, `condition_value` reflects the total actual cost of executing the statement, in milliseconds. • If `condition` is RELATIVE_COST_DIFFERENCE, `condition_value` reflects the cost of executing, as a percentage of the estimated cost.
enabled	BIT	Whether the row is enabled. That is, whether the tracing settings in the row are active. 1 is enabled; 0 is disabled.

Related Information

Diagnostic Tracing Conditions (Deprecated)
Diagnostic Tracing Scopes (Deprecated)
SQL Anywhere Profiler
ATTACH TRACING Statement (Deprecated) [page 775]
REFRESH TRACING LEVEL Statement (Deprecated) [page 1327]

1.5.1.1.3 RowGenerator Table (dbo)

The dbo.RowGenerator table is provided as a read-only table that has 255 rows. This table can be useful for queries which produce small result sets and which need a range of numeric values.

The RowGenerator table is used by system procedures and views, and should not be modified in any way.

You can also use the sa_rowgenerator system procedure to generate a range of numeric values.

Column name	Column type
row_num	SMALLINT

row_num

A value between 1 and 255.

Related Information

sa_rowgenerator System Procedure [page 1690]

1.6 System Procedures

There are hundreds of system in the software, many of which are for internal use only. The documentation explains the system procedures that are for external use.

All system procedures can be enabled and disabled as secure features.

In this section:

Viewing Details About System Procedures and Functions [page 1512]
Access the definitions for system procedures and functions from *SQL Central*.

Web Services System Procedures [page 1512]
Use these system procedures when working with web services.

Roles and Privileges System Procedures [page 1513]
Use these procedures when administering roles and privileges.

MAPI and SMTP System Procedures [page 1513]
System procedures for sending electronic mail using the Microsoft Messaging API standard (MAPI) or the Internet standard Simple Mail Transfer Protocol (SMTP) are included. These system procedures are implemented as extended system procedures: each procedure calls a function in an external DLL.

Directory and File System Procedures [page 1516]
You can access the local file structure of the computer running a database server by using directory access servers, or by using file and directory system procedures, such as the sp_create_directory system procedure

Secured Feature System Procedures [page 1517]

There are several system procedures provided for administering features that can be secured.

Adaptive Server Enterprise System and Catalog Procedures [page 1518]
SQL Anywhere has implemented support for a subset of the Adaptive Server Enterprise system and catalog procedures. However, for information about using these procedures, refer to your Adaptive Server Enterprise documentation.

Alphabetical List of System Procedures [page 1520]
These are the system procedures that you can use to return data or perform operations on data.

1.6.1 Viewing Details About System Procedures and Functions

Access the definitions for system procedures and functions from *SQL Central*.

Procedure

1. Use the SQL Anywhere 17 plug-in to connect to the database.
2. Right-click the database and then click *Configure Owner Filter*.
3. Click *DBO* and then click *OK*.
4. In the left pane, double-click *Procedures & Functions*.
5. In the left pane, select the procedure and in the right pane click the *SQL* tab.

Results

The procedure or function definition appears on the *SQL* tab.

1.6.2 Web Services System Procedures

Use these system procedures when working with web services.

- sa_http_header_info system procedure
- sa_http_php_page system procedure
- sa_http_php_page_interpreted system procedure
- sa_http_variable_info system procedure
- sa_set_http_header system procedure
- sa_set_http_option system procedure
- sa_set_soap_header system procedure

Related Information

Web Services Functions [page 221]
The Database Server as an HTTP Web Server
-xs Database Server Option
sa_http_header_info System Procedure [page 1613]
sa_http_php_page System Procedure [page 1615]
sa_http_php_page_interpreted System Procedure [page 1616]
sa_http_variable_info System Procedure [page 1618]
sa_set_http_header System Procedure [page 1711]
sa_set_http_option System Procedure [page 1713]
sa_set_soap_header System Procedure [page 1718]

1.6.3 Roles and Privileges System Procedures

Use these procedures when administering roles and privileges.

- sp_displayroles system procedure
- sp_has_role system procedure
- sp_objectpermission system procedure
- sp_proc_priv system procedure
- sp_sys_priv_role_info system procedure

Related Information

sp_displayroles System Procedure [page 1768]
sp_has_role System Procedure [page 1782]
sp_objectpermission System Procedure [page 1796]
sp_proc_priv System Procedure [page 1805]
sp_sys_priv_role_info System Procedure [page 1836]

1.6.4 MAPI and SMTP System Procedures

System procedures for sending electronic mail using the Microsoft Messaging API standard (MAPI) or the Internet standard Simple Mail Transfer Protocol (SMTP) are included. These system procedures are implemented as extended system procedures: each procedure calls a function in an external DLL.

These procedures are owned by the dbo role. Users must be granted the EXECUTE privilege on these system procedures before they can use them.

To use the MAPI or SMTP system procedures, a MAPI or SMTP email system must be accessible from the database server computer.

The MAPI and SMTP system procedures are:

xp_startmail

Starts a mail session in a specified mail account by logging on to the MAPI message system.

xp_startsmtp

Starts a mail session in a specified mail account by logging on to the SMTP message system.

xp_sendmail

Sends a mail message to specified users.

xp_stopmail

Closes the MAPI mail session.

xp_stopsmtp

Closes the SMTP mail session.

xp_get_mail_error_code

Returns information about the most recent SMTP or MAPI error.

xp_get_mail_error_text

Returns the text of the most recent SMTP error.

Example

The following procedure notifies a set of people that a backup has been completed. It does not check return codes.

```
CREATE PROCEDURE notify_backup( )
BEGIN
    CALL xp_startmail(  mail_user='ServerAccount',
                        mail_password='ServerPassword' );
    CALL xp_sendmail( recipient='IS Group',
                      subject='Backup',
                      "message"='Backup completed' );
    CALL xp_stopmail( );
END;
```

The following procedure notifies a set of people that a backup has been completed and includes error checking.

```
CREATE OR REPLACE PROCEDURE notify_backup_with_error_check( )
BEGIN
    DECLARE result_code INTEGER;
    DECLARE error_code INTEGER;
    DECLARE error_text LONG VARCHAR;
    SET result_code = xp_startmail(  'ServerAccount', -- mail_user
                                     'ServerPassword' -- mail_password
                                  );
    IF result_code = 0 THEN
         SET result_code = xp_sendmail( 'IS Group', --recipient
                                        'Backup', -- subject
                                        NULL, -- cc_recipient
                                        NULL, -- bcc_recipient
```

```
                                        NULL, -- query
                                        'Backup completed' -- message
                                      );
    ENDIF;
    IF result_code = 0 THEN
        SET result_code = xp_stopmail( );
    ENDIF;
    IF result_code = 0 THEN
        MESSAGE 'Backup completed message successfully sent';
    ELSE
        SET error_code = xp_get_mail_error_code();
        SET error_text = xp_get_mail_error_text();
        MESSAGE 'Error:' || result_code ||
                ' Code:' || error_code ||
                ' Text:' || error_text;
    ENDIF;
END;;
```

In this section:

Related Information

xp_startmail System Procedure [page 1880]
xp_startsmtp System Procedure [page 1881]
xp_sendmail System Procedure [page 1874]
xp_stopmail System Procedure [page 1885]
xp_stopsmtp System Procedure [page 1886]
xp_get_mail_error_code System Procedure [page 1865]
xp_get_mail_error_text System Procedure [page 1867]

1.6.4.1 Status Codes for MAPI and SMTP System Procedures

The following codes can be returned by either the MAPI or the SMTP system procedures:

Status code	Meaning
-1	Unknown error[1]
0	Success
1	An invalid parameter was supplied
2	Out of memory
3	xp_startmail or xp_startsmtp was not called
4	Bad host name

Status code	Meaning
5	Connect error[1]
6	Secure connection error[1]
7	MAPI functions are not available[1]

[1] Use the xp_get_mail_error_code and xp_get_mail_error_text system procedures to get additional information about a return code.

Related Information

xp_startmail System Procedure [page 1880]
xp_startsmtp System Procedure [page 1881]
xp_sendmail System Procedure [page 1874]
xp_stopmail System Procedure [page 1885]
xp_stopsmtp System Procedure [page 1886]
xp_get_mail_error_code System Procedure [page 1865]
xp_get_mail_error_text System Procedure [page 1867]

1.6.5 Directory and File System Procedures

You can access the local file structure of the computer running a database server by using directory access servers, or by using file and directory system procedures, such as the sp_create_directory system procedure

The file and directory system procedures are easier to use than directory access servers, but they are not as flexible nor as powerful as directory access proxy tables and servers.

Use the following system procedures to manipulate directories and files on the computer where a server resides:

sp_list_directory system procedure

Lists the contents of a specified directory.

sp_create_directory system procedure

Creates a specified directory.

sp_copy_directory system procedure

Copies specified directory to another location.

sp_move_directory system procedure

Moves a specified directory.

sp_delete_directory system procedure

Deletes a specified directory.

sp_copy_file system procedure

Copies a specified file.

sp_move_file system procedure

Moves a specified file

sp_delete_file system procedure

Deletes a specified file.

The following system procedures can also be used to read and write to files:

xp_read_file system procedure

Reads a file and returns the contents of the file as a LONG BINARY variable.

xp_write_file system procedure

Writes data to a file from a SQL statement.

sp_disk_info system procedure Returns disk drive information for a specified file or directory.

> **i** Note
>
> If disk sandboxing is enabled, locations required by the system procedures must be inside the sandbox for the procedures to work.

Related Information

Directory Access Servers
sp_copy_directory System Procedure [page 1755]
sp_copy_file System Procedure [page 1757]
sp_create_directory System Procedure [page 1758]
sp_delete_directory System Procedure [page 1764]
sp_delete_file System Procedure [page 1765]
sp_list_directory System Procedure [page 1786]
sp_move_directory System Procedure [page 1793]
sp_move_file System Procedure [page 1795]
xp_read_file System Procedure [page 1871]
xp_write_file System Procedure [page 1887]

1.6.6 Secured Feature System Procedures

There are several system procedures provided for administering features that can be secured.

- sa_server_option system procedure
- sp_alter_secure_feature_key system procedure
- sp_create_secure_feature_key system procedure
- sp_drop_secure_feature_key system procedure
- sp_list_secure_feature_keys system procedure
- sp_use_secure_feature_key system procedure

Related Information

sa_server_option System Procedure [page 1698]
sp_alter_secure_feature_key System Procedure [page 1752]
sp_create_secure_feature_key System Procedure [page 1760]
sp_drop_secure_feature_key System Procedure [page 1771]
sp_list_secure_feature_keys System Procedure [page 1791]
sp_use_secure_feature_key System Procedure [page 1854]

1.6.7 Adaptive Server Enterprise System and Catalog Procedures

SQL Anywhere has implemented support for a subset of the Adaptive Server Enterprise system and catalog procedures. However, for information about using these procedures, refer to your Adaptive Server Enterprise documentation.

In this section:

Adaptive Server Enterprise System Procedures [page 1518]
SQL Anywhere includes some Adaptive Server Enterprise system procedures.

Adaptive Server Enterprise Catalog Procedures [page 1519]
SQL Anywhere implements a subset of the Adaptive Server Enterprise catalog procedures.

1.6.7.1 Adaptive Server Enterprise System Procedures

SQL Anywhere includes some Adaptive Server Enterprise system procedures.

While these procedures perform the same functions as they do in Adaptive Server Enterprise, they are not identical. If you have preexisting scripts that use these procedures, you may want to review the procedure code. To see the text of a stored procedure, run the following command.

```
sp_helptext 'dbo.procedure_name'
```

The implemented system procedures are described in the following table.

System procedure name / parameters	Description
sp_addgroup `@grpname`	Adds a group to a database
sp_addlogin `@login_name` , `@passwd` [, `@defaultdb` [, `@deflanguage` [, `@fullname`]]]	Adds a new login ID to a database
sp_addmessage `@message_num` , `@message_text` [, `@language`]	Adds a user-defined message to ISYSUSERMESSAGE, for use by stored procedure PRINT and RAISERROR calls
sp_addtype `@typename` , `@phystype` [, `@ident_null`]	Creates a user-defined data type

System procedure name / parameters	Description
sp_adduser `@login_name` [, `@name_in_db` [, `@grpname`]]	Adds a new user ID to a database
sp_changegroup `@grpname` , `@name_in_db`	Changes a user group or adds a user to a group
sp_dropgroup `@grpname`	Drops a group from a database
sp_droplogin `@login_name`	Drops a login ID from a database
sp_dropmessage `@message_number` [, `@language`]	Drops a user-defined message
sp_droptype `@typename`	Drops a user-defined data type
sp_dropuser `@name_in_db`	Drops a user ID from a database
sp_getmessage `@message_num` , `@msg_var` [, `@language`]	Retrieves a stored message string from ISYSUSERMESSAGE, for PRINT and RAISERROR statements.
sp_helptext [`@objname`]	Displays the text of a system procedure, trigger, or view
sp_password `@caller_pswd` , `@new_pswd` [, `@login_name`]	Adds or changes a password for a user ID

1.6.7.2 Adaptive Server Enterprise Catalog Procedures

SQL Anywhere implements a subset of the Adaptive Server Enterprise catalog procedures.

While these procedures perform the same functions as they do in Adaptive Server Enterprise, they are not identical. If you have preexisting scripts that use these procedures, you may want to review the procedure code. To see the text of a stored procedure, run the following command.

```
sp_helptext 'dbo.procedure_name'
```

The implemented catalog procedures are described in the following table.

Catalog procedure name / parameters	Description
sp_column_privileges	Unsupported
sp_columns [`@table_name` [, `@table_owner` [, `@table_qualifier` [, `@column_name`]]]]	Returns the data types of the specified columns
sp_fkeys [`@pktable_name` [, `@pktable_owner` [, `@pktable_qualifier` [, `@fktable_name` [, `@fktable_owner` [, `@fktable_qualifier`]]]]]]	Returns foreign key information about the specified table
sp_pkeys `@table_name` [, `@table_owner` [, `@table_qualifier`]]	Returns primary key information about the specified table
sp_special_columns `@table_name` [, `@table_owner` [, `@table_qualifier` [, `@col_type`]]]	Returns the optimal set of columns that uniquely identify a row in the specified table
sp_sproc_columns `@sp_name` [, `@sp_owner` [, `@sp_qualifier` [, `@column_name`]]]	Returns information about the input and return parameters of a stored procedure
sp_statistics [`@table_name` [, `@table_owner` [, `@table_qualifier` [, `@index_name` [, `@is_unique`]]]]]	Returns information about tables and their indexes

Catalog procedure name / parameters	Description
sp_stored_procedures [`@sp_name` [, `@sp_owner` [, `@sp_qualifier`]]]	Returns information about one or more stored procedures
sp_tables [`@table_name` [, `@table_owner` [, `@table_qualifier` [, `@table_type`]]]	Returns a list of objects that can appear in a FROM clause

1.6.8 Alphabetical List of System Procedures

These are the system procedures that you can use to return data or perform operations on data.

A few system procedures, such as sa_get_table_definition, are implemented as functions. However, because they are used in the same context and manner as system procedures, they are included with the system procedures, and their naming is similar to the system procedures (sa_xxx).

In this section:

Returns a summary of request timing information for all connections when the database server has request timing logging enabled.

sa_performance_statistics System Procedure [page 1671]
Returns performance statistics for the server, databases, and connections.

sa_post_login_procedure System Procedure [page 1672]
Determines whether a user's password is about to expire.

sa_procedure_profile System Procedure [page 1674]
Reports information about the execution time for each line within procedures, functions, events, or triggers that have been executed in a database.

sa_procedure_profile_summary System Procedure [page 1677]
Reports summary information about the execution times for all procedures, functions, events, or triggers that have been executed in a database.

sa_recompile_views System Procedure [page 1680]
Locates view definitions stored in the catalog that do not have column definitions and causes the column definitions to be created.

sa_refresh_materialized_views System Procedure [page 1681]
Initializes all materialized views that are in an uninitialized state.

sa_refresh_text_indexes System Procedure [page 1683]
Refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH.

sa_remove_tracing_data System Procedure (Deprecated) [page 1684]
The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The sa_remove_tracing_data system procedure permanently deletes from the diagnostic tracing tables all records pertaining to the specified logging (tracing) session ID.

sa_report_deadlocks System Procedure [page 1685]
Retrieves information about deadlocks from an internal buffer created by the database server.

sa_reserved_words System Procedure [page 1687]
Returns a list of reserved words. Many of the keywords that appear in SQL statements are reserved words.

sa_reset_identity System Procedure [page 1688]
Allows the next identity value to be set for a table. Use this procedure to change the AUTOINCREMENT value for the next row that will be inserted.

sa_rowgenerator System Procedure [page 1690]
Returns a result set with rows between a specified start and end value.

sa_save_trace_data System Procedure [page 1693]
Saves tracing data to base tables.

sa_send_udp System Function [page 1694]
Sends a UDP packet to the specified address.

sa_server_messages System Procedure [page 1695]
Allows you to return messages from the database server messages window as a result set.

sa_server_option System Procedure [page 1698]
Overrides a database server option while the database server is running.

sa_set_http_header System Procedure [page 1711]
Permits a web service to set an HTTP response header.

Validates the password of the current user.

sp_alter_secure_feature_key System Procedure [page 1752]
Alters a previously defined secured feature key by modifying the authorization code and/or feature list.

sp_auth_sys_role_info System Procedure [page 1754]
Returns the mapping of authorities from previous versions of SQL Anywhere to their corresponding compatibility roles.

sp_copy_directory System Procedure [page 1755]
Copies a directory to a specified location.

sp_copy_file System Procedure [page 1757]
Copies a file to a specified location.

sp_create_directory System Procedure [page 1758]
Creates a directory on the computer.

sp_create_secure_feature_key System Procedure [page 1760]
Creates a new secured feature key.

sp_db_cache_contents System Procedure [page 1762]
Returns one row per dbspace with a bitmap showing which pages are currently in the cache.

sp_delete_directory System Procedure [page 1764]
Deletes the specified directory.

sp_delete_file System Procedure [page 1765]
Deletes the specified file from the computer.

sp_disk_info System Procedure [page 1766]
Returns disk drive information for a given path.

sp_displayroles System Procedure [page 1768]
Returns all roles granted to the specified system privilege, system role, user-defined role, or user name, or displays the entire hierarchy tree of roles.

sp_drop_secure_feature_key System Procedure [page 1771]
Deletes a secured feature key.

sp_find_top_statements System Procedure [page 1773]
Reports performance statistics for each logged statement and plan combination.

sp_forward_to_remote_server System Procedure [page 1776]
Allows an application to execute a SQL statement on a remote server and retrieve any result sets generated by that statement.

sp_generate_key_pair System Procedure [page 1777]
Creates a new RSA encryption key pair.

sp_get_last_synchronize_result System Procedure [page 1779]
Returns information about the last synchronization initiated by the SYNCHRONIZE statement.

sp_has_role System Procedure [page 1782]
Returns whether the invoker of the procedure has been granted the specified system privilege or user-defined role.

sp_http_listeners System Procedure [page 1784]
Lists the HTTP and HTTPS connection listeners used for the specified database.

sp_list_directory System Procedure [page 1786]

Returns information about the files and subdirectories in a specified directory.

sp_list_mutexes_semaphores System Procedure [page 1788]
Returns information about all temporary and permanent mutexes and semaphores, including which connection is holding each mutex and whether a semaphore is being waited for.

sp_list_secure_feature_keys System Procedure [page 1791]
Returns a list of defined secured feature keys.

sp_login_environment System Procedure [page 1792]
Sets connection options when users log in.

sp_move_directory System Procedure [page 1793]
This function moves the directory pointed to by source_path to the destination pointed to by destination_path.

sp_move_file System Procedure [page 1795]
Moves the specified file to a new directory on the computer.

sp_objectpermission System Procedure [page 1796]
Generates a report on object privileges granted to a specified role or user ID, or a report on object privileges granted on a specified object or dbspace.

sp_parse_json System Procedure [page 1799]
Returns a representation of JSON data using SQL data types.

sp_plancache_contents System Procedure [page 1802]
Returns the contents of a plan cache for a connection, including statistics indicating the average, minimum, and maximum execution times, and the number of times cursors have been built for a statement.

sp_proc_priv System Procedure [page 1805]
Returns the list of system privileges required to run a system procedure.

sp_property_history System Procedure [page 1807]
Returns values for all database server properties tracked by the database.

sp_read_db_pages System Procedure [page 1810]
Reads the specified pages into the cache.

sp_read_etd System Procedure [page 1813]
Reads the specified event trace data (ETD) file and returns the contents of the file as a set of rows.

sp_remote_columns System Procedure [page 1816]
Produces a list of the columns in a remote table, and a description of their data types.

sp_remote_exported_keys System Procedure [page 1818]
Provides information about tables with foreign keys on a specified primary table.

sp_remote_imported_keys System Procedure [page 1820]
Provides information about remote tables with primary keys that correspond to a specified foreign key.

sp_remote_pcols System Procedure [page 1822]
Produces a list of the columns in a remote table, and a description of their data types.

sp_remote_primary_keys System Procedure [page 1825]
Provides primary key information about remote tables using remote data access.

sp_remote_procedures System Procedure [page 1827]
Returns a list of the procedures on a remote server.

xp_read_file System Procedure [page 1871]
Reads a file and returns the contents of the file as a LONG BINARY variable.

xp_scanf System Procedure [page 1873]
Extracts substrings from an input string using a format string.

xp_sendmail System Procedure [page 1874]
Sends an email message to the specified recipients once a session has been started with xp_startmail or xp_startsmtp. The procedure accepts messages of any length.

xp_sprintf System Procedure [page 1879]
Builds a result string from a set of input strings.

xp_startmail System Procedure [page 1880]
Starts an email session under MAPI.

xp_startsmtp System Procedure [page 1881]
Starts an email session under SMTP.

xp_stopmail System Procedure [page 1885]
Closes a MAPI email session.

xp_stopsmtp System Procedure [page 1886]
Closes an SMTP email session.

xp_write_file System Procedure [page 1887]
Writes data to a file from a SQL statement.

1.6.8.1 sa_ansi_standard_packages System Procedure

Returns information about the non-core SQL extensions used in a SQL statement.

Syntax

```
sa_ansi_standard_packages(
standard
, statement
)
```

Parameters

standard

Use this LONG VARCHAR parameter to specify the standard to use for the core extensions. One of SQL:1999 or SQL:2003.

statement

Use this LONG VARCHAR parameter to specify the SQL statement to evaluate.

Result Set

Column name	Data type	Description
package_id	VARCHAR(10)	The feature identifier.
package_name	LONG VARCHAR	The feature name.

Remarks

If there are no non-core extensions used for the statement, the result set is empty.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

Following is an example call to the sa_ansi_standard_packages system procedure:

```
CALL sa_ansi_standard_packages( 'SQL:2003',
'SELECT *
   FROM ( SELECT o.SalesRepresentative,
   o.Region,
                SUM( s.Quantity * p.UnitPrice ) AS total_sales,
                DENSE_RANK() OVER ( PARTITION BY o.Region,
                                    GROUPING( o.SalesRepresentative )
                                    ORDER BY total_sales DESC ) AS sales_rank
            FROM Product p, SalesOrderItems s, SalesOrders o
            WHERE p.ID = s.ProductID AND s.ID = o.ID
            GROUP BY GROUPING SETS( ( o.SalesRepresentative, o.Region ),
o.Region ) ) AS DT
   WHERE sales_rank <= 3
   ORDER BY Region, sales_rank');
```

The example generates the following result set:

package_id	package_name
T612	Advanced OLAP operations

package_id	package_name
T611	Elementary OLAP operations
F591	Derived tables
T431	Extended grouping capabilities

Related Information

The Embedded SQL Preprocessor
SQLFLAGGER Function [Miscellaneous] [page 563]
sql_flagger_error_level Option
sql_flagger_warning_level Option

1.6.8.2 sa_audit_string System Procedure

Adds a string to the transaction log.

Syntax

```
sa_audit_string( string )
```

Parameters

string

The VARCHAR(128) string of characters to add to the transaction log.

Remarks

If auditing is turned on, this system procedure adds a comment to the auditing information stored in the transaction log. The string can be a maximum of 128 characters.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE AUDITING system privilege.

Side Effects

None

Example

The following example uses sa_audit_string to add a comment to the transaction log:

```
CALL sa_audit_string( 'Auditing test' );
```

Related Information

Database Activity Audits
auditing Option

1.6.8.3 sa_certificate_info System Procedure

Displays information about the specified certificate that is stored in the database.

Syntax

```
sa_certificate_info( cert_name )
```

Parameters

cert_name

The CHAR(128) certificate name that was used in the CREATE CERTIFICATE statement.

Result Set

Column name	Data type	Description
name	CHAR(128)	The name of the attribute.
value	LONG VARCHAR	The value of the attribute.

Remarks

If cert_name is NULL or no certificate in the ISYSCERTIFICATE table has that name, no rows are returned. Otherwise, the following keys are always returned:

name	value
Name	Value of the cert_name column in ISYSCERTIFICATE.
ID	Value of the object_id column in ISYSCERTIFICATE.

Other rows returned have keys that correspond to attributes of the certificate such as Common Name, Country Code, Locality, and Organization.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns information about the certificate mycert:

```
CALL sa_certificate_info( 'mycert' );
```

Related Information

CREATE CERTIFICATE Statement [page 820]
SYSCERTIFICATE System View [page 1900]

1.6.8.4 sa_char_terms System Procedure

Breaks a CHAR string into terms and returns each term as a row along with its position.

Syntax

```
sa_char_terms(
text
[, config_name
[, owner ] ]
)
```

Parameters

text

The LONG VARCHAR string you are parsing.

config_name

Use this optional CHAR(128) parameter to specify the text configuration object to apply when processing the string. The default value is 'default_char'.

owner

Use this optional CHAR(128) parameter to specify the owner of the text configuration object. The default value is NULL. The current user is assumed if the owner is not specified or if NULL is specified.

Remarks

You can use this system procedure to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following statement returns the terms in the CHAR string "It's a work-at-home day!" using the default CHAR text configuration object, default_char:

```
CALL sa_char_terms ('It's a work-at-home day!', 'default_char', 'sys');
```

term	position
It	1
s	2
a	3
work	4
at	5
home	6
day	7

Related Information

Full Text Search
Text Configuration Object Concepts and Reference
sa_nchar_terms System Procedure [page 1664]

1.6.8.5 sa_check_commit System Procedure

Checks for outstanding referential integrity violations before a commit.

Syntax

```
sa_check_commit(
tname
, keyname
)
```

Parameters

tname

A VARCHAR(128) parameter containing the name of a table with a row that is currently violating referential integrity.

keyname

A VARCHAR(128) parameter containing the name of the corresponding foreign key index.

Remarks

If the database option wait_for_commit is On, or if a foreign key is defined using CHECK ON COMMIT in the CREATE TABLE statement, you can update the database and cause a referential integrity violation if the violations are resolved before the changes are committed.

You can use the sa_check_commit system procedure to check whether there are any outstanding referential integrity violations before attempting to commit your changes.

The returned parameters indicate the name of a table containing a row that is currently violating referential integrity, and the name of the corresponding foreign key index.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following set of statements can be executed from Interactive SQL. Rows are deleted from the Departments table in the sample database and a referential integrity violation occurs. The call to the sa_check_commit system procedure checks which tables and keys have outstanding violations, and the rollback cancels the change.

```
SET TEMPORARY OPTION wait_for_commit='On';
DELETE FROM Departments;
CREATE OR REPLACE VARIABLE tname VARCHAR( 128 );
CREATE OR REPLACE VARIABLE keyname VARCHAR( 128 );
CALL sa_check_commit( tname, keyname );
SELECT tname, keyname;
ROLLBACK;
SET TEMPORARY OPTION wait_for_commit='Off';
```

Related Information

wait_for_commit Option

CREATE TABLE Statement [page 1002]

1.6.8.6 sa_clean_database System Procedure

Starts the database cleaner and sets the maximum length of time for which it can run.

Syntax

```
sa_clean_database( [ duration ] )
```

Parameters

duration

Use this optional UNSIGNED INTEGER parameter to specify the number of seconds that the clean operation is allowed to run. The default is 0 which is interpreted to mean that no limit is imposed on the duration that the cleaner runs.

Remarks

The database cleaner is an internal task that runs on a default schedule. You can use this system procedure to force the database cleaner to run immediately and to specify how long the cleaner can run each time it is invoked. When the duration is 0, the database cleaner runs until all pages in all dbspaces have been cleaned.

If you use this system procedure to start the database cleaner while a database is being validated, the database cleaner does not run until validation is complete.

Some database tasks, such as processing snapshot isolation transactions, index maintenance, and deleting rows, can execute more efficiently if some portions of the request are deferred to a later time. These deferrable activities typically involve cleanup by removing deleted, historical, and otherwise unnecessary entries from database pages, or reorganizing database pages for more efficient access.

Postponing some of these activities not only allows the current request to finish more quickly, it potentially allows cleanup to occur when the database server is less active. These unnecessary entries are identified so that they are not visible to other transactions; however, they do take up space on a page, and must be removed at some point.

The database cleaner performs any deferred cleanup activities. It is scheduled to run every 20 seconds. When it is invoked, the database cycles sequentially through the database's dbspaces, examining and cleaning each cleanable page before moving on to the next one. When invoked automatically by the database server, the database cleaner is a self-tuning process. The amount of work that the database cleaner performs, and the duration for which it executes, depend on several factors, including the fraction of outstanding cleanable pages in a dbspace, the current amount of activity in the database server, and the amount of time that the database cleaner has already spent cleaning. If, after running for 0.5 seconds, the cleaner detects active requests in the server, it stops and reschedules itself to execute at its regular interval. The database cleaner attempts to

process pages when there are no other requests executing in the server, and therefore takes advantage of periods of server inactivity.

Database cleaner statistics are available through four database properties:

CleanablePagesAdded

returns the number of pages that need to be cleaned

CleanablePagesCleaned

returns the number of pages that have already been cleaned

CleanableRowsAdded

returns the number of rows that need to be cleaned

CleanableRowsCleaned

returns the number of rows that have already been cleaned

The difference between the values of CleanablePagesAdded and CleanablePagesCleaned indicates how many database pages still require cleaning.

You can use the sa_clean_database system procedure to configure the database cleaner to run until all the pages in a database are cleaned, or to specify a maximum duration for the database cleaner to run.

To further customize the behavior of the database cleaner, you can set up an event that starts the database cleaner if the number of pages or rows that need to be cleaned exceed a specified threshold.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

The following example sets the duration of the database cleaner to 10 seconds:

```
CALL sa_clean_database( 10 );
```

The following example creates a scheduled event that runs daily to allow the database cleaner to run until all pages in the database are cleaned:

```
CREATE EVENT DailyDatabaseCleanup
SCHEDULE
   START TIME '6:00 pm'
```

```
   ON ( 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday' )
   HANDLER
      BEGIN
         CALL sa_clean_database( );
      END;
```

The following example forces the database cleaner to run when 20% or more of the pages in the database need to be cleaned:

```
CREATE EVENT PeriodicCleaner
SCHEDULE
BETWEEN '9:00 am' and '5:00 pm'
EVERY 1 HOURS
HANDLER
BEGIN
     DECLARE @num_db_pages INTEGER;
     DECLARE @num_dirty_pages INTEGER;
     -- Get the number of database pages
     SELECT (SUM( DB_EXTENDED_PROPERTY( 'FileSize', t.dbspace_id ) -
                  DB_EXTENDED_PROPERTY( 'FreePages', t.dbspace_id ) ))
     INTO @num_db_pages
     FROM (SELECT dbspace_id FROM SYSDBSPACE) AS t;
     -- Get the number of pages to be cleaned
     SELECT (DB_PROPERTY( 'CleanablePagesAdded' ) -
                  DB_PROPERTY( 'CleanablePagesCleaned' ))
     INTO @num_dirty_pages;
     -- Check whether the number of dirty pages exceeds 20% of
     -- the size of the database
     IF @num_dirty_pages > @num_db_pages * 0.20 THEN
       -- Start cleaning the database for a maximum of 60 seconds
       CALL sa_clean_database( 60 );
     END IF;
END;
```

Related Information

CREATE EVENT Statement [page 847]

1.6.8.7 sa_column_stats System Procedure

Returns various statistics about the specified column(s). These statistics are not related to the column statistics maintained for use by the optimizer.

Syntax

```
sa_column_stats(
[ tab_name
[, col_name
[, tab_owner
[, max_rows ] ] ] ]
)
```

Parameters

tab_name

This optional CHAR(128) parameter specifies the name of the table. If this parameter is not specified, statistics are calculated for all columns in all table(s). The default is '%'.

col_name

This optional CHAR(128) parameter specifies the columns for which to calculate statistics. If this parameter is not specified, statistics are calculated for all columns in the specified table(s). The default is '%'.

tab_owner

This optional CHAR(128) parameter specifies the owner of the table. If this parameter is not specified, the database server uses the owner of the first table that matches the `tab_name` specified. The default is '%'.

max_rows

This optional INTEGER parameter specifies the number of rows to use for the calculations. The default is 1000. Specifying 0 instructs the database server to calculate the ratio based on all the rows in the table.

Result Set

With the exception of table_owner, table_name, and column_name, all values in the result set are NULL for non-string columns. Also, for empty tables, num_rows_processed and num_values_compressed are 0, while all other values are NULL.

Column name	Data type	Description
table_owner	CHAR(128)	The owner of the table.
table_name	CHAR(128)	The table name.
column_name	CHAR(128)	The column name.
num_rows_processed	INTEGER	The total number of rows read to calculate the statistics.
num_values_compressed	INTEGER	The number of values in the column that are compressed. If the column is not compressed, the value is 0.
avg_compression_ratio	DOUBLE	The average compression ratio, expressed as a percentage reduction in size, for compressed values in the column. If the column is not compressed, the value is NULL.
avg_length	DOUBLE	The average length of all non-NULL strings in the column.
stddev_length	DOUBLE	The standard deviation of the lengths of all non-NULL strings in the column.
min_length	INTEGER	The minimum length of non-NULL strings in the column.

Column name	Data type	Description
max_length	INTEGER	The maximum length of strings in the column.
avg_uncompressed_length	DOUBLE	The average length of all uncompressed, non-NULL strings in the column.
stddev_uncompressed_length	DOUBLE	The standard deviation of the lengths of all uncompressed, non-NULL strings in the column.
min_uncompressed_length	INTEGER	The minimum length of all uncompressed, non-NULL strings in the column.
max_uncompressed_length	INTEGER	The maximum length of all uncompressed, non-NULL strings in the column.

Remarks

The database server determines the columns that match the owner, table, and column names specified, and then for each one, calculates statistics for the data in each specified column. By default, the database server only uses the first 1000 rows of data.

For avg_compression_ratio, values cannot be greater than, or equal to 100, however, they can be less than 0 if highly incompressible data (for example, data that is already compressed) is inserted into a compressed column. Higher values indicate better compression. For example, if the number returned is 80, then the size of the compressed data is 80% less than the size of the uncompressed data.

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must be the owner of table, or have SELECT privilege on the columns, or have the MONITOR or MANAGE ANY STATISTICS system privilege.

> **i** Note
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

In this example, you use the sa_column_stats system procedure in a SELECT statement to determine which columns in the database are benefiting most from column compression:

```
SELECT * FROM sa_column_stats()
  WHERE num_values_compressed > 0
  ORDER BY avg_compression_ratio desc;
```

In this example, you narrow your selection from the previous example to tables owned by bsmith:

```
SELECT * FROM sa_column_stats( tab_owner='GROUPO' )
  WHERE num_values_compressed > 0
  ORDER BY avg_compression_ratio desc;
```

Related Information

Column Compression Considerations

1.6.8.8 sa_conn_activity System Procedure

Returns the most recently prepared SQL statement for each connection to the indicated database on the server.

Syntax

```
sa_conn_activity( [ connidparm ] )
```

Parameters

connidparm

Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Name	VARCHAR(255)	Returns the name of the current connection. Temporary connection names have `INT:` prepended to the connection name.
Userid	VARCHAR(255)	Returns the user ID for the connection.
DBNumber	INTEGER	Returns the ID number of the database.
LastReqTime	VARCHAR(255)	Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.
LastStatement	LONG VARCHAR	Returns the most recently prepared SQL statement for the current connection.

Remarks

If `connidparm` is less than zero, then information for the current connection is returned. If `connidparm` is not supplied or is NULL, then information is returned for all connections to all databases running on the database server.

The sa_conn_activity system procedure returns a result set consisting of the most recently prepared SQL statement for the connection. Recording of statements must be enabled for the database server before calling sa_conn_activity. To do this, specify the -zl option when starting the database server, or execute the following:

```
CALL sa_server_option('RememberLastStatement','ON');
```

This procedure is useful when the database server is busy and you want to obtain information about the last SQL statement prepared for each connection. This feature can be used as an alternative to request logging.

Privileges

You must have EXECUTE privilege on the system procedure.

To obtain a list of all connection IDs, you must also have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side Effects

None

Example

The following example uses the sa_conn_activity system procedure to display the most recently prepared SQL statement for each connection.

```
CALL sa_conn_activity( );
```

Number	Name	Userid	DBNumber	...
1,949	SQL_DBC_117acc40	DBA	0	...
1,948	setup	User1	0	...
...	...	...	...	...

Related Information

-zl Database Server Option
sa_server_option System Procedure [page 1698]
List of Connection Properties

1.6.8.9 sa_conn_compression_info System Procedure

Summarizes communication compression rates.

Syntax

```
sa_conn_compression_info( [ connidparm ] )
```

Parameters

connidparm

Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result Set

Column name	Data type	Description
Type	VARCHAR(20)	Returns a string identifying whether the compression statistics that follow represent either one connection (Connection), or all connections to the server (Server).
ConnNumber	INTEGER	Returns an INTEGER representing a connection ID number. Returns NULL if the Type is Server.
Compression	VARCHAR(10)	Returns On or Off to indicate whether communication compression is enabled on the connection. Returns NULL if the Type is Server, or ON/OFF if the Type is Connection.
TotalBytes	INTEGER	Returns an INTEGER representing the total number of actual bytes both sent and received.
TotalBytesUnComp	INTEGER	Returns an INTEGER representing the number of bytes that would have been sent and received if compression were disabled.
CompRate	NUMERIC(5,2)	Returns a NUMERIC (5,2) value representing the overall compression rate. For example, a value of 0 indicates that no compression occurred. A value of 75 indicates that the data was compressed by 75%, or down to one quarter of its original size.
CompRateSent	NUMERIC(5,2)	Returns a NUMERIC (5,2) value representing the compression rate for data sent to the client.
CompRateReceived	NUMERIC(5,2)	Returns a NUMERIC (5,2) value representing the compression rate for data received from the client.
TotalPackets	INTEGER	Returns an INTEGER representing the total number of actual packets both sent and received.
TotalPacketsUnComp	INTEGER	Returns an INTEGER representing the total number of packets that would have been sent and received if compression was disabled.

Column name	Data type	Description
CompPktRate	NUMERIC(5,2)	Returns a NUMERIC (5,2) value representing the overall compression rate of packets.
CompPktRateSent	NUMERIC(5,2)	Returns a NUMERIC (5,2) value representing the compression rate of packets sent to the client.
CompPktRateReceived	NUMERIC(5,2)	Returns a NUMERIC (5,2) value representing the compression rate of packets received from the client.

Remarks

If `connidparm` is less than zero, then a result set consisting of compression properties for the current connection is returned. If `connidparm` is not supplied or is NULL, then compression properties are returned for all connections to all databases running on the database server.

Privileges

You must have EXECUTE privilege on the system procedure.

To obtain a list of all connection IDs, you must also have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side Effects

None

Example

The following example uses the sa_conn_compression_info system procedure to return a result set summarizing compression properties for all connections to the server.

```
CALL sa_conn_compression_info( );
```

Type	ConnNumber	Compression	TotalBytes	...
Connection	79	Off	7841	...
Server	(NULL)	(NULL)	2737761	...

Type	ConnNumber	Compression	TotalBytes	...
...	...	...	...	...

Related Information

List of Connection Properties

1.6.8.10 sa_conn_info System Procedure

Reports connection property information.

Syntax

```
sa_conn_info( [ connidparm ] )
```

Parameters

connidparm

This optional INTEGER parameter specifies the connection ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Name	VARCHAR(255)	Returns an identifier (string) for the current connection. Temporary connection names have **INT:** prepended to the connection name.
Userid	VARCHAR(255)	Returns the user ID for the connection.
DBNumber	INTEGER	Returns the ID number of the database.

Column name	Data type	Description
LastReqTime	VARCHAR(255)	Returns the time at which the last request for the specified connection started, in the timezone of the database. This property can return an empty string for internal connections, such as events.
ReqType	VARCHAR(255)	Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CONNECT_POOL_CACHE.
CommLink	VARCHAR(255)	Returns the communication link for the connection. This is one of the network protocols supported by SQL Anywhere, or local for a same-computer connection.
NodeAddr	VARCHAR(255)	Returns the address of the client in a client/server connection.
ClientPort	INTEGER	Returns the client's TCP/IP port number or 0 if the connection isn't a TCP/IP connection.
ServerPort	INTEGER	Returns the database server's TCP/IP port number or 0.
BlockedOn	INTEGER	Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.
LockRowID	UNSIGNED BIGINT	Returns the identifier of the locked row. LockRowID is NULL if the connection is not waiting on a lock associated with a row (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated row).

Column name	Data type	Description
LockIndexID	INTEGER	Returns the identifier of the locked index. LockIndexID is -1 if the lock is associated with all indexes on the table in LockTable. LockIndexID is NULL if the connection is not waiting on a lock associated with an index (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated index).
LockTable	VARCHAR(255)	Returns the name of the table associated with a lock if the connection is currently waiting for a lock. The LockTable value is an empty string if there is no lock, or if the object associated with the lock is not a table.
UncommitOps	INTEGER	Returns the number of uncommitted operations.
ParentConnection	INTEGER	Returns the connection ID of the connection that created a temporary connection to perform a database operation (such as performing a backup or creating a database). For other types of connections, this property returns NULL.
LockObject	VARCHAR(255)	Returns the name of the object associated with the lock for which the connection is waiting, if any. If the object is a table, this value is the same as the LockTable value. LockObject is NULL if the connection is not waiting for a lock.
LockObjectType	CHAR(20)	Returns the type of the object associated with the lock for which the connection is waiting, if any. LockObjectType is NULL if the connection is not waiting for a lock.

Remarks

If `connidparm` is less than zero, then a result set consisting of connection properties for the current connection is returned. If `connidparm` is not supplied or is NULL, then connection properties are returned for all connections to all databases running on the database server.

In a block situation, the BlockedOn value returned by this procedure allows you to check which users are blocked, and who they are blocked on. The sa_locks system procedure can be used to display the locks held by the blocking connection.

For more information based on any of these properties, you can execute something similar to the following:

```
SELECT *, DB_NAME( DBNumber ),
   CONNECTION_PROPERTY( 'LastStatement', Number )
   FROM sa_conn_info( );
```

The value of LockRowID can be used to look up a lock in the output of the sa_locks procedure.

The value in LockIndexID can be used to look up a lock in the output of the sa_locks procedure. Also, the value in LockIndexID corresponds to the primary key of the ISYSIDX system table, which can be viewed using the SYSIDX system view.

Every lock has an associated table, so the value of LockTable can be used to unambiguously determine whether a connection is waiting on a lock.

Privileges

You must have EXECUTE privilege on the system procedure.

To obtain a list of all connection IDs, you must also have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side Effects

None

Example

The following example uses the sa_conn_info system procedure to return a result set summarizing connection properties for all connections to the server.

```
CALL sa_conn_info( );
```

Number	Name	Userid	DBNumber	...
79	SQL_DBC_10dcf810	DBA	0	...
46	setup	User1	0	...
...	...	...	...	...

The following example uses the sa_conn_info system procedure to return a result set showing which connection created a temporary connection.

```
SELECT Number, Name, ParentConnection FROM sa_conn_info();
```

Connection 8 created the temporary connection that executed a CREATE DATABASE statement.

```
Number      Name                ParentConnection
-------------------------------------------------
1000000048  INT: CreateDB       8
9           SQL_DBC_14675af8    (NULL)
8           SQL_DBA_152d5ac0    (NULL)
```

The following example uses the sa_conn_info system to return the number of blocked connections.

```
SELECT COUNT(*) FROM sa_conn_info()
WHERE blockedOn = connection_property('number');
```

Related Information

List of Connection Properties
sa_locks System Procedure [page 1633]
SYSIDX System View [page 1916]

1.6.8.11 sa_conn_list System Procedure

Returns a result set containing connection IDs.

Syntax

```
sa_conn_list(
[ connidparm
[, dbidparm ] ]
)
```

Parameters

connidparm

Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

dbidparm

Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.

Remarks

If `connidparm` is greater than zero, then information for the supplied connection is returned. If `connidparm` is less than zero, then information for the current connection is returned. If `connidparm` and `dbidparm` are not supplied or are NULL, then connection IDs for all connections to all databases running on the database server are returned.

If `connidparm` is NULL and `dbidparm` is greater than or equal to zero, then connection IDs for only that database are returned. If `connidparm` is NULL and `dbidparm` is less than zero, then connection IDs for just the current database are returned.

Privileges

You must have EXECUTE privilege on the system procedure.

To obtain a list of all connection IDs, you must also have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side Effects

None

Example

The following example uses the sa_conn_list system procedure to display a list of connection IDs.

```
CALL sa_conn_list( );
```

Number
1,949
1,948

Number
...

Related Information

sa_db_list System Procedure [page 1565]
sa_conn_options System Procedure [page 1553]

1.6.8.12 sa_conn_options System Procedure

Returns property information for connection properties that correspond to database options.

Syntax

```
sa_conn_options( [ connidparm ] )
```

Parameters

connidparm

Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
PropNum	INTEGER	Returns the connection property number.
OptionName	VARCHAR(255)	Returns the option name.
OptionDescription	VARCHAR(255)	Returns the option description.
Value	LONG VARCHAR	Returns the option value.

Remarks

Returns the connection ID as Number, and the PropNum, OptionName, OptionDescription, and Value for each available connection property that corresponds to a database option.

If `connidparm` is less than zero, then option values for the current connection are returned. If `connidparm` is not supplied or is NULL, then option values are returned for all connections to the current database provided that you have the required system privileges.

Privileges

You must have EXECUTE privilege on the system procedure.

To obtain the options for all connections, you must also have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege. If you do not have one of these system privileges, then the results returned are for the current connection only.

Side Effects

None

Example

The following example uses the sa_conn_options system procedure to display property information for connection properties that correspond to database options.

```
SELECT * FROM sa_conn_options()
    ORDER BY OptionName;
```

Number	PropNum	OptionName	OptionDescription	Value
1,952	502	blocking	Controls response to locking conflicts	On
1,952	503	blocking_timeout	Controls the time a transaction waits to obtain a lock	0
...	...	...	...	...

Related Information

Database Options

sa_db_list System Procedure [page 1565]
sa_conn_list System Procedure [page 1551]
List of Connection Properties

1.6.8.13 sa_conn_properties System Procedure

Reports connection property information.

Syntax

```
sa_conn_properties( [ connidparm ] )
```

Parameters

connidparm

Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
PropNum	INTEGER	Returns the connection property number.
PropName	VARCHAR(255)	Returns the connection property name.
PropDescription	VARCHAR(255)	Returns the connection property description.
Value	LONG VARCHAR	Returns the connection property value.

Remarks

Returns the connection ID as Number, and the PropNum, PropName, PropDescription, and Value for each available connection property. Values are returned for all connection properties, database option settings related to connections, and statistics related to connections. Valid properties with NULL values are also returned.

If `connidparm` is less than zero, then property values for the current connection are returned. If `connidparm` is not supplied or is NULL, then property values are returned for all connections to the current database.

Privileges

You must have EXECUTE privilege on the system procedure.

To obtain a list of all connection IDs, you must also have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side Effects

None

Example

The following example uses the sa_conn_properties system procedure to return a result set summarizing connection property information for all connections.

```
CALL sa_conn_properties( );
```

Number	PropNum	PropName	...
79	37	ClientStmtCacheHits	...
79	38	ClientStmtCacheMisses	...
...	...	...	...

This example uses the sa_conn_properties system procedure to return a list of all connections, in decreasing order by CPU time*:

```
SELECT Number AS connection_number,
    CONNECTION_PROPERTY ( 'Name', Number ) AS connection_name,
    CONNECTION_PROPERTY ( 'Userid', Number ) AS user_id,
  CAST ( Value AS NUMERIC ( 30, 2 ) ) AS approx_cpu_time
  FROM sa_conn_properties( )
  WHERE PropName = 'ApproximateCPUTime'
  ORDER BY approx_cpu_time DESC;
```

Related Information

System Functions [page 224]
sa_conn_list System Procedure [page 1551]

sa_conn_options System Procedure [page 1553]
List of Connection Properties

1.6.8.14 sa_convert_ml_progress_to_timestamp System Procedure

For MobiLink scripted uploads only. This converts the progress value for scripted upload from an UNSIGNED BIGINT to a TIMESTAMP.

Syntax

```
sa_convert_ml_progress_to_timestamp( progress )
```

Parameters

progress

Use this UNSIGNED BIGINT parameter to specify the progress value to convert to a TIMESTAMP.

Returns

The function returns the TIMESTAMP that is represented by the value passed in.

Remarks

This function is the inverse of sa_convert_timestamp_to_ml_progress.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example converts an UNSIGNED BIGINT value to a timestamp value (2009-10-20 13:36:51.199).

```
SELECT sa_convert_ml_progress_to_timestamp( 3465034611199 );
```

Related Information

Scripted Upload
sa_convert_timestamp_to_ml_progress System Procedure [page 1558]

1.6.8.15 sa_convert_timestamp_to_ml_progress System Procedure

For MobiLink scripted uploads only. This converts the progress value for scripted upload from a TIMESTAMP to an UNSIGNED BIGINT.

Syntax

```
sa_convert_timestamp_to_ml_progress( t1 )
```

Parameters

t1

Use this TIMESTAMP parameter to specify the progress value to convert to an UNSIGNED BIGINT.

Returns

The function returns an UNSIGNED BIGINT that represents the timestamp passed in as a parameter.

Remarks

This procedure is the inverse of sa_convert_ml_progress_to_timestamp.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following examples convert timestamp values to UNSIGNED BIGINT values.

```
SELECT sa_convert_timestamp_to_ml_progress( CURRENT TIMESTAMP );
```

```
SELECT sa_convert_timestamp_to_ml_progress( '2009-10-20 13:36:51.199' );
```

Related Information

Scripted Upload
sa_convert_ml_progress_to_timestamp System Procedure [page 1557]

1.6.8.16 sa_copy_cursor_to_temp_table System Procedure

Creates a temporary table and copies the result set of an open cursor to it.

Syntax

```
sa_copy_cursor_to_temp_table(
cursor_name
, table_name
[, first_row
[, max_rows ] ]
)
```

Parameters

cursor_name

Use this VARCHAR(256) parameter to specify the name of the open cursor.

table_name

Use this VARCHAR(256) parameter to specify the name of the temporary table.

first_row

Use this BIGINT parameter to specify the number of the first row to copy to the temporary table. The default is 1.

max_rows

Use this BIGINT parameter to specify the maximum number of rows to copy to the temporary table. The default is 9223372036854775807 (all rows).

Remarks

Suppose you have a cursor of several integer columns. sa_copy_cursor_to_temp_table creates a temporary table using a statement in this form:

```
BEGIN
  CREATE LOCAL TEMPORARY TABLE TempTab (
      col1 INT,
      col2 INT,
  ...
rownum bigint primary key )
END;
```

sa_copy_cursor_to_temp_table names the columns col1,col2, and so on to avoid duplication of names or difficulty if cursor columns do not have a well defined name (for example, if they are a complex expression).

Once the temporary table is created, the contents of the open cursor are inserted by moving to the row number indicated by `first_row`, and inserting the number of rows indicated by `max_rows`. After the contents have been inserted into the temporary table, the cursor is re-positioned at its original location.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

Copying from the cursor fetches the rows using the cursor's isolation settings. This may acquire locks on rows and have other effects equivalent to fetching from the cursor.

If concurrent changes are made outside of the current connection and the cursor is not protected from these by materialization or isolation settings, then it is possible that the cursor will be positioned on a different row after the procedure completes. For example, if the previous current row of the cursor was deleted, the cursor could be repositioned on the row after the original position.

If an error occurs while copying from the cursor, the cursor enters an invalid state, and further operations on the cursor fail with an error.

Example

The following batch creates a cursor named myCursor and loads it with data from the Products table. The cursor is then opened (OPEN statement). A DROP statement drops myTempTable, if it already exists. Calling sa_copy_cursor_to_temp_table creates a temporary table called myTempTable and copies the contents of myCursor into it. Finally, a SELECT statement returns the data that was copied into the temporary table from the cursor:

```
BEGIN
  DECLARE myCursor CURSOR FOR
      SELECT ID, Name, Description, Color, Quantity FROM Products;
  OPEN myCursor;
  DROP TABLE IF EXISTS myTempTable;
  CALL sa_copy_cursor_to_temp_table( 'myCursor', 'myTempTable' );
  CLOSE myCursor;
  SELECT * FROM myTempTable;
END
```

Related Information

sa_list_cursors System Procedure [page 1629]
sa_describe_cursor System Procedure [page 1573]

1.6.8.17 sa_cpu_topology System Procedure

Returns the processor topology of the computer that the database server is running on.

Syntax

sa_cpu_topology( )

Result Set

Column name	Data type	Description
os_id	UNSIGNED INTEGER	Returns the logical processor identifier used by the underlying operating system. For example, on Windows this value matches the processor ID that appears in the Windows Task Manager.
socket	UNSIGNED INTEGER	Returns a CPU socket or package identifier.
core	UNSIGNED INTEGER	Returns an identifier for a core within a specific socket.
thread	UNSIGNED INTEGER	Returns an identifier for a thread within a specific socket and core.
apic	UNSIGNED INTEGER	Returns an identifier for the logical processor within the system. It contains the encoding of the socket, core, and thread.
"group"	UNSIGNED INTEGER	Returns the processor group number as assigned by the operating system on Windows. On Solaris, it is the processor set identifier. Otherwise, it is 0.
numa_node	UNSIGNED INTEGER	Returns the NUMA node assigned by the operating system on Windows.
online	BIT	Returns 1 if and only if the processor is online.
in_use	BIT	Returns 1 if the logical processor described by the current row is online and enabled in the process affinity mask for the database server process, and 0 otherwise.
user_selected	BIT	Indicates which physical processors were specified using the -gta database server option or the ProcessorAffinity system procedure.

Remarks

On 32-bit and 64-bit Intel x86/x64 platforms, the information returned by this procedure accurately describes the underlying hardware. On all other platforms, there is one row per logical processor in the system, but the column values are set as follows:

- os_id, socket, and apic are equal
- core, thread, and numa_node are zero
- group is zero, except on Solaris where it is the pset identifier

The results may not be accurate on some virtual machines.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns a result set containing the processor topology of the computer that the database server is running on.

```
CALL sa_cpu_topology( );
```

Related Information

-gta Database Server Option
sa_server_option System Procedure [page 1698]

1.6.8.18 sa_db_info System Procedure

Reports database property information.

Syntax

```
sa_db_info( [ dbidparm ] )
```

Parameters

dbidparm

Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Alias	VARCHAR(255)	Returns the database name.
File	VARCHAR(255)	Returns the file name of the database root file, including path.
ConnCount	INTEGER	Returns the number of connections to the database. The property value does not include connections used for internal operations, but it does include connections used for events and external environment support.
PageSize	INTEGER	Returns the page size of the database, in bytes.
LogName	VARCHAR(255)	Returns the file name of the transaction log, including path.

Remarks

If you specify a database ID, sa_db_info returns a single row containing the Number, Alias, File, ConnCount, PageSize, and LogName for the specified database.

If `dbidparm` is greater than zero, then properties for the supplied database are returned. If `dbidparm` is less than zero, then properties for the current database are returned. If `dbidparm` is not supplied or is NULL, then properties for all databases running on the database server are returned.

Privileges

You must have EXECUTE privilege on the system procedure.

To execute this system procedure for other databases, you must also have either the SERVER OPERATOR or MONITOR system privilege.

Side Effects

None

Example

The following statement returns a row for each database that is running on the server:

```
CALL sa_db_info( );
```

Related Information

sa_db_properties System Procedure [page 1569]
List of Database Server Properties

1.6.8.19 sa_db_list System Procedure

Returns a database ID.

Syntax

```
sa_db_list( [ dbidparm ] )
```

Parameters

dbidparm

Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	The database ID number.

Remarks

If `dbidparm` is greater than zero, then the ID for the supplied database is returned. If `dbidparm` is less than zero, then the ID for the current database is returned. If `dbidparm` is not supplied or is NULL, then IDs for all databases running on the database server are returned.

Privileges

You must have EXECUTE privilege on the system procedure.

To execute this system procedure for other databases, you must also have either the SERVER OPERATOR or MONITOR system privilege.

Side Effects

None

Example

The following example returns the number of databases running on a database server.

```
SELECT count(*) FROM sa_db_list();
```

The following example uses the system procedure sa_db_list and the function DB_Name to return a list of the databases running on a database server.

```
SELECT DB_NAME(Number) FROM sa_db_list();
```

Related Information

sa_conn_list System Procedure [page 1551]
sa_conn_options System Procedure [page 1553]
DB_NAME Function [System] [page 336]

1.6.8.20 sa_db_option System Procedure

Overrides a database option while the database is running.

Syntax

```
sa_db_option(
opt
, val
)
```

Parameters

opt

Use this CHAR(128) parameter to specify a database option name.

val

Use this LONG VARCHAR parameter to specify the new value for the database option.

Remarks

Database administrators can use this procedure to override some database options temporarily, without restarting the database.

The option values that are changed using this procedure are reset to their default values when the database shuts down. To change an option value every time the database is started, specify the corresponding database option when the database is started (if one exists).

The following option settings can be changed:

Option name	Values	System privilege	Additional information
DiskSandbox	ON, **OFF**	SERVER OPERATOR	When DiskSandbox is set to ON, it restricts read-write file operations on the database to the directory where the main database file is located and any subdirectories of this directory. To use the sa_db_option system procedure to change disk sandbox settings, you must provide the secured feature key for the manage_disk_sandbox feature.

Option name	Values	System privilege	Additional information
PropertyHistoryList	comma-delimited list of database server properties	MANAGE PROPERTY HISTORY	When PropertyHistoryList is turned on, a default list of properties is selected. You can specify a comma-delimited list of database server properties to track. The default is an empty list. If this option is not set then only properties tracked by the database server or by other databases on the same database server can be queried. If the memory required to track all specified properties is greater than the limit set by the PropertyHistorySize database server option or by the available memory, then an error is raised and the property list is not modified for the database.
CollectStmtPerfStats	ON, **OFF**	MONITOR	When CollectStmtPerfStats is set to ON, it collects statement performance summary information. When it is set to OFF, it stops data collection and discards data that has been collected.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None.

Example

For the following example to work, the database server must be started with the option `-sk securefkey`.

This example enables the SYSTEM secured feature key that includes MANAGE_KEYS, creates a new secured feature key called SECURITY with case-sensitive authorization code NewSecurityCode, and then uses the new secured feature key to enable the DiskSandbox option.

```
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
CALL sp_create_secure_feature_key( 'security', 'NewSecurityCode',
'manage_security' );
CALL sp_use_secure_feature_key( 'security', 'NewSecurityCode' );
CALL sa_db_option( 'DiskSandbox', 'on' );
```

Related Information

-sbx Database Server Option
disk_sandbox Option
sa_server_option System Procedure [page 1698]

1.6.8.21 sa_db_properties System Procedure

Reports database property information.

Syntax

```
sa_db_properties( [ dbidparm ] )
```

Parameters

dbidparm

Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result Set

Column name	Data type	Description
Number	INTEGER	The database ID number.

Column name	Data type	Description
PropNum	INTEGER	The database property number.
PropName	VARCHAR(255)	The database property name.
PropDescription	VARCHAR(255)	The database property description.
Value	LONG VARCHAR	The database property value.

Remarks

If you specify a database ID, the sa_db_properties system procedure returns the database ID number and the PropNum, PropName, PropDescription, and Value for each available database property. Values are returned for all database properties and statistics related to databases. Valid properties with NULL values are also returned.

If `dbidparm` is greater than zero, then database properties for the supplied database are returned. If `dbidparm` is less than zero, then database properties for the current database are returned. If `dbidparm` is not supplied or is NULL, then database properties for all databases running on the database server are returned.

Privileges

You must have EXECUTE privilege on the system procedure.

To execute this system procedure for other databases, you must also have either the SERVER OPERATOR or MONITOR system privilege.

Side Effects

None

Example

The following example uses the sa_db_properties system procedure to return a result set summarizing database properties for all databases when the invoker has SERVER OPERATOR or MONITOR system privilege. Otherwise, database properties for the current database are returned.

```
CALL sa_db_properties( );
```

Number	PropNum	PropName	...
0	0	ConnCount	...

Number	PropNum	PropName	...
0	1	IdleCheck	...
0	2	IdleWrite	...
...	...	...	...

The following example uses the sa_db_properties system procedure to return a result set summarizing database properties for a second database.

```
CALL sa_db_properties( 1 );
```

Related Information

sa_db_info System Procedure [page 1563]
List of Database Server Properties

1.6.8.22 sa_dependent_views System Procedure

Returns the list of all dependent views for a given table or view.

Syntax

```
sa_dependent_views(
[ tbl_name
[, owner_name ] ]
)
```

Parameters

tbl_name

Use this optional CHAR(128) parameter to specify the name of the table or view. The default is NULL.

owner_name

Use this optional CHAR(128) parameter to specify the owner for `tbl_name`. The default is NULL.

Result Set

Column name	Data type	Description
table_id	UNSIGNED INTEGER	The object ID of the table or view.
dep_view_id	UNSIGNED INTEGER	The object ID of the dependent views.

Remarks

Use this procedure to obtain the list of IDs of tables and their dependent views.

No errors are generated if no existing tables satisfy the specified criteria for table and owner names. The following conditions also apply:

- If both `owner` and `tbl_name` are NULL, information is returned on all tables that have dependent views.
- If `tbl_name` is NULL but `owner` is specified, information is returned on all tables owned by the specified owner.
- If `tbl_name` is specified but `owner` is NULL, information is returned on any one of the tables with the specified name.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

In this example, the sa_dependent_views system procedure is used to obtain the list of IDs for the views that are dependent on the SalesOrders table. The procedure returns the table_id for SalesOrders, and the dep_view_id for the dependent view, ViewSalesOrders.

```
CALL sa_dependent_views( 'SalesOrders' );
```

In this example, the sa_dependent_views system procedure is used in a SELECT statement to obtain the list of names of views dependent on the SalesOrders table. The procedure returns the ViewSalesOrders view.

```
SELECT t.table_name FROM SYSTAB t,
sa_dependent_views( 'SalesOrders' ) v
```

```
WHERE t.table_id = v.dep_view_id;
```

Related Information

View Dependencies
SYSDEPENDENCY System View [page 1905]

1.6.8.23 sa_describe_cursor System Procedure

Describes the name and type information for the columns of a cursor.

Syntax

```
sa_describe_cursor( cursor_name )
```

Parameters

cursor_name

This VARCHAR(256) value identifies the open cursor to describe.

Result Set

Column name	Data type	Description
column_number	INTEGER	The ordinal position of the column described by this row, starting at 1.
name	VARCHAR(128)	The name of the column.
domain_id	SMALLINT	The data type of the column.
domain_name	VARCHAR(128)	The data type name of the column.
domain_name_with_size	VARCHAR(160)	The data type name, including size and precision (as used in CREATE TABLE or CAST functions).
width	INTEGER	The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.

Column name	Data type	Description
scale	INTEGER	The number of digits after the decimal point for numeric data type columns, and zero for all other data types.
declared_width	INTEGER	The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.
user_type_id	SMALLINT	The user-defined data type if applicable, otherwise NULL.
user_type_name	VARCHAR(128)	The user-defined data type if applicable, otherwise NULL.
correlation_name	VARCHAR(128)	The correlation name associated with the expression if applicable, otherwise NULL.
base_table_id	UNSIGNED INTEGER	The table_id if the expression is a column, otherwise NULL.
base_column_id	UNSIGNED INTEGER	The column_id if the expression is a column, otherwise NULL.
base_owner_name	VARCHAR(128)	The owner name if the expression is a column, otherwise NULL.
base_table_name	VARCHAR(128)	The table name if the expression is a column, otherwise NULL.
base_column_name	VARCHAR(128)	The column name if the expression is a column, otherwise NULL.
nulls_allowed	BIT	The indicator whether the expression can be NULL (1).
is_autoincrement	BIT	An indicator whether the expression is an AUTOINCREMENT column (1).
is_key_column	BIT	An indicator whether the expression is part of a key for the result set (1). For more information, see the Remarks section below.
is_added_key_column	BIT	An indicator whether the expression is an added key column (1). For more information, see the Remarks section below.

Remarks

The sa_describe_cursor system procedure provides an API-independent mechanism for retrieving the description of the columns returned by the cursor. The system procedure can be useful when writing stored procedures that work with dynamic SQL.

The sa_describe_cursor system procedure can be used in a CALL statement or in the FROM clause of a SELECT statement.

`cursor_name` must refer to an open cursor in the current connection. Use the sa_list_cursors system procedure to get the list of open cursors for the connection.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following batch creates a cursor named myCursor on the Products table and then opens it. The sa_describe_cursor system procedure is used to describe the columns of the cursor. A result set containing 5 rows, one for each column, is produced.

```
BEGIN
  DECLARE myCursor CURSOR FOR
      SELECT ID, Name, Description, Color, Quantity FROM Products;
  OPEN myCursor;
  CALL sa_describe_cursor( 'myCursor' );
  CLOSE myCursor;
END
```

Related Information

sa_list_cursors System Procedure [page 1629]
sa_copy_cursor_to_temp_table System Procedure [page 1559]
SYSDOMAIN System View [page 1905]
SYSUSER System View [page 1965]
SYSTABCOL System View [page 1953]
SYSUSERTYPE System View [page 1968]
SYSDOMAIN System View [page 1905]

1.6.8.24 sa_describe_query System Procedure

Describes the result set for a query with one row describing each output column of the query.

Syntax

```
sa_describe_query(
query
[, add_keys ]
)
```

Parameters

query

Use this LONG VARCHAR parameter to specify the text of the SQL statement being described.

add_keys

Use this optional BIT parameter to specify whether to determine a set of columns that uniquely identify rows in the result set for the query being described. The default is 0; the database server does not attempt to identify the columns. See the Remarks section below for a full explanation of this parameter.

Result Set

Column name	Data type	Description
column_number	INTEGER	The ordinal position of the column described by this row, starting at 1.
name	VARCHAR(128)	The name of the column.
domain_id	SMALLINT	The data type of the column.
domain_name	VARCHAR(128)	The data type name.
domain_name_with_size	VARCHAR(160)	The data type name, including size and precision (as used in CREATE TABLE or CAST functions).
width	INTEGER	The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.
scale	INTEGER	The number of digits after the decimal point for numeric data type columns, and zero for all other data types.

Column name	Data type	Description
declared_width	INTEGER	The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.
user_type_id	SMALLINT	The type_id of the user-defined data type if there is one, otherwise NULL.
user_type_name	VARCHAR(128)	The name of the user-defined data type if there is one, otherwise NULL.
correlation_name	VARCHAR(128)	The correlation name associated with the expression if one is available, otherwise NULL.
base_table_id	UNSIGNED INTEGER	The table_id if the expression is a column, otherwise NULL.
base_column_id	UNSIGNED INTEGER	The column_id if the expression is a column, otherwise NULL.
base_owner_name	VARCHAR(128)	The owner name if the expression is a column, otherwise NULL.
base_table_name	VARCHAR(128)	The table name if the expression is a column, otherwise NULL.
base_column_name	VARCHAR(128)	The column name if the expression is a column, otherwise NULL.
nulls_allowed	BIT	An indicator that is 1 if the expression can be NULL, otherwise 0.
is_autoincrement	BIT	An indicator that is 1 if the expression is a column declared to be AUTOINCREMENT, otherwise 0.
is_key_column	BIT	An indicator that is 1 if the expression is part of a key for the result set, otherwise 0. For more information, see the Remarks section below.
is_added_key_column	BIT	An indicator that is 1 if the expression is an added key column, otherwise 0. For more information, see the Remarks section below.

Remarks

The sa_describe_query procedure provides an API-independent mechanism to describe the name and type information for the expressions in the result set of a query.

When 1 is specified for `add_keys`, the sa_describe_query procedure attempts to find a set of columns from the objects being queried that, when combined, can be used as a key to uniquely identify rows in result set of the query being described. The key takes the form of one or more columns from the objects being queried, and may include columns that are not explicitly referenced in the query. If the optimizer finds a key, the column or

columns used in the key are identified in the results by an is_key_column value of 1. If no key is found, an error is returned.

For any column that is included in the key but that is not explicitly referenced in the query, the is_added_key_column value is set to 1 to indicate that the column has been added to the results for the procedure; otherwise, the value of is_added_key_column is 0.

If you do not specify `add_keys`, or you specify a value of 0, the optimizer does not attempt to find a key for the result set, and the is_key_column and is_added_key_column columns contain NULL.

The declared_width and width values both describe the size of a column. The declared_width describes the size of the column as defined by the CREATE TABLE statement or by the query, while the width value depends on the data type.

The client representation of a type may be different from the database server. For example, DATE, TIME, and TIMESTAMP data types are converted to strings if the return_date_time_as_string option is on. The TIMESTAMP WITH TIME ZONE data type is always formatted as a string when sent to the client. The width value for these types is based solely on the length of corresponding format option string and may or may not reflect that actual formatted length. For example, specifying more than 6 digits of precision for the fractional seconds part will affect the width value, but the actual formatted string will have at most 6 digits of precision.

For CHARACTER data types, columns declared with character-length semantics have a declared_width value that matches the CREATE TABLE size, while the width value gives the maximum number of bytes needed to store the returned string.

For NUMERIC, DECIMAL, FLOAT, REAL, and DOUBLE data types, declared_width and width are identical. The width does not represent the length of the formatted string which may include sign and decimal point indicators. For numeric/decimal values, width represents the precision. For floating-point values, width represents the storage size in bytes.

The following table provides some examples.

Declaration	width	declared_width
CHAR(10)	10	10
CHAR(10 CHAR)	40	10
DATE	depends on the length of the date_format option string	8
DOUBLE	8	8
REAL	4	4
NUMERIC(10, 3)	10 (precision)	10 (precision)
TIME	depends on the length of the time_format option string	8
TIMESTAMP	depends on the length of the timestamp_format option string	8
TIMESTAMP WITH TIME ZONE	depends on the length of the timestamp_with_time_zone_format option string	8

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example describes the information returned when querying all columns in the Departments table:

```
SELECT *
FROM sa_describe_query( 'SELECT * FROM Departments DEPT' );
```

The results show the values of the is_key_column and is_added_key_column as NULL because the `add_keys` parameter was not specified.

The following example describes the information returned by querying the DepartmentName and Surname columns of the Employees table, joined with the Departments table:

```
SELECT  *
FROM  sa_describe_query( 'SELECT DepartmentName, Surname
 FROM Employees E JOIN Departments D ON E.EmployeeID = D.DepartmentHeadId',
 add_keys = 1 );
```

The results shows a 1 in rows 3 and 4 of the result set, indicating that the columns needed to uniquely identify rows in the result set for the query are Employees.EmployeeID and Departments.DepartmentID. Also, a 1 is present in the is_added_key_column for rows 3 and 4 because Employees.EmployeeID and Departments.DepartmentID were not explicitly referenced in the query being described.

Related Information

Character Data Types [page 130]
EXPRTYPE Function [Miscellaneous] [page 379]
return_date_time_as_string Option
SYSDOMAIN System View [page 1905]
SYSDOMAIN System View [page 1905]
SYSUSERTYPE System View [page 1968]
SYSTAB System View [page 1950]
SYSUSER System View [page 1965]

1.6.8.25 sa_describe_shapefile System Procedure

Describes the names and types of columns contained in an ESRI shapefile. This system feature is for use with the spatial data features.

Syntax

```
sa_describe_shapefile(
shp_filename
, srid
[, encoding ]
)
```

Parameters

shp_filename

A VARCHAR(512) parameter that identifies the location of the ESRI shapefile. The file name must have the extension `.shp` and must have an associated `.dbf` file with the same base name located in the same directory. The path is relative to the database server, not the client application.

srid

An INTEGER parameter that identifies the SRID for the geometries in the shapefile. Specify NULL to indicate the column can store multiple SRIDs. Specifying NULL limits the operations that can be performed on the geometry values.

encoding

An optional VARCHAR(50) parameter that identifies the encoding to use when reading the shapefile. The default is NULL. When encoding is NULL, the ISO-8859-1 character set is used.

Result Set

Column name	Data type	Description
column_number	INTEGER	The ordinal position of the column described by this row, starting at 1.
name	VARCHAR(128)	The name of the column.
domain_name_with_size	VARCHAR(160)	The data type name, including size and precision (as used in CREATE TABLE or CAST functions).

Remarks

The sa_describe_shapefile system procedure is used to describe the name and type of columns in an ESRI shapefile. This information can be used to create a table to load data from a shapefile using the LOAD TABLE or

INPUT statements. Alternately, this system procedure can be used to read a shapefile by specifying the WITH clause for OPENSTRING...FORMAT SHAPEFILE.

Privileges

You must have EXECUTE privilege on the system procedure. Additionally:

- If the -gl database option is set to DBA, you must have one of the following system privileges:
  - ALTER ANY TABLE
  - ALTER ANY OBJECT
  - LOAD ANY TABLE
  - READ FILE
- If the -gl database option is set to ALL, no privileges are required.
- If the -gl database option is set to NONE, you must have the READ FILE system privilege.

Example

The following example displays a string that was used to create a table for storing shapefile data:

```
BEGIN
    DECLARE create_cmd LONG VARCHAR;
    SELECT 'create table if not exists esri_load( record_number int primary key,
' ||
         (SELECT list( name || ' ' || domain_name_with_size, ', ' ORDER BY
column_number )
     FROM sa_describe_shapefile( 'c:\\esri\\tgr36069trt00.shp', 1000004326 )
     WHERE column_number > 1 ) || ' )'
     INTO create_cmd;
     SELECT create_cmd;
     EXECUTE IMMEDIATE create_cmd;
END
```

You can load the shapefile data into the table using the following statement (provided that you have the LOAD ANY TABLE system privilege and that the -gl database option has not been set to NONE):

```
LOAD TABLE esri_load
USING FILE 'c:\\esri\\tgr36069trt00.shp'
FORMAT SHAPEFILE;
```

Related Information

LOAD TABLE Statement [page 1247]
INPUT Statement [Interactive SQL] [page 1224]
Support for ESRI Shapefiles

1.6.8.26 sa_disable_auditing_type System Procedure

Disables auditing of specific events.

Syntax

```
sa_disable_auditing_type( types )
```

Parameters

types

Use this VARCHAR(128) parameter to specify a comma-delimited string containing one or more of the following values:

all

disables all types of auditing.

connect

disables auditing of both successful and failed connection attempts.

connectFailed

disables auditing of failed connection attempts.

DDL

disables auditing of DDL statements.

options

disables auditing of public options.

permission

disables auditing of permission checks, user checks, and SETUSER statements.

permissionDenied

disables auditing of failed permission and user checks.

triggers

disables auditing in response to trigger events.

xp_cmdshell disables auditing for xp_cmdshell.

Remarks

Use sa_disable_auditing_type to specify which types of auditing to exclude. This system procedure removes the specified events from the current set of audit events. Use sa_enable_auditing_type to add events to the current set of audit events. These system procedures set the PUBLIC *auditing_options* database option so the setting is permanent.

Set the PUBLIC *auditing* database option to On or Off to enable or disable auditing.

By default, all events are audited (types='all'). If you want a smaller set, use the sa_disable_auditing_type system procedure to clear the events you are not interested in; or use the sa_disable_auditing_type system procedure to clear all events and then use the sa_enable_auditing_type system procedure to specify which types of auditing you want.

If the set of events is empty and you set the PUBLIC *auditing* database option to On, no auditing information is recorded. To re-establish auditing, you must use the sa_enable_auditing_type system procedure to specify which types of information you want to audit.

If you set the PUBLIC *auditing* database option to Off, then no auditing information is recorded.

Specify the location where events are logged with the *audit_log* database option.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SET ANY SECURITY OPTION system privilege.

Side Effects

None

Example

The following example disables all auditing:

```
CALL sa_disable_auditing_type( 'all' );
```

The following example enables only DDL and triggers auditing:

```
CALL sa_disable_auditing_type( 'all' );
CALL sa_enable_auditing_type( 'DDL,triggers' );
```

The following example enables all auditing except for DDL and options auditing:

```
CALL sa_enable_auditing_type( 'all' );
CALL sa_disable_auditing_type( 'DDL,options' );
```

Related Information

Database Activity Audits
sa_enable_auditing_type System Procedure [page 1585]
auditing Option

1.6.8.27 sa_disk_free_space System Procedure

Reports information about space available for a dbspace, transaction log, transaction log mirror, and/or temporary file.

Syntax

```
sa_disk_free_space( [ p_dbspace_name ] )
```

Parameters

p_dbspace_name

Use this VARCHAR(128) parameter to specify the name of a dbspace, transaction log file, transaction log mirror file, or temporary file. The default is NULL.

If there is a dbspace called log, mirror, or temp, you can prefix the keyword with an underscore. For example, use _log to get information about the transaction log file if a dbspace called log exists.

Specify SYSTEM to get information about the main database file, TEMPORARY or TEMP to get information about the temporary file, TRANSLOG to get information about the transaction log, or TRANSLOGMIRROR to get information about the transaction log mirror.

Result Set

Column name	Data type	Description
dbspace_name	VARCHAR(128)	This is the dbspace name, transaction log file, transaction log mirror file, or temporary file.
free_space	UNSIGNED BIGINT	The number of free bytes on the volume.
total_space	UNSIGNED BIGINT	The total amount of disk space available on the drive where the dbspace resides.

Remarks

If the `p_dbspace_name` parameter is not specified or is NULL, then the result set contains one row for each dbspace, plus one row for each of the transaction log, transaction log mirror, and temporary file, if they exist. If `p_dbspace_name` is specified, then exactly one or zero rows are returned (zero if no such dbspace exists, or if log or mirror is specified and there is no log or mirror file).

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY DBSPACE system privilege.

Side Effects

None

Example

The following example uses the sa_disk_free_space system procedure to return a result set containing information about available space.

```
CALL sa_disk_free_space( );
```

dbspace_name	free_space	total_space
system	10952101888	21410402304
translog	10952101888	21410402304
temporary	10952101888	21410402304

Related Information

Predefined Dbspaces

1.6.8.28 sa_enable_auditing_type System Procedure

Specifies which events to include in auditing.

Syntax

```
sa_enable_auditing_type( types )
```

Parameters

types

Use this VARCHAR(128) parameter to specify a comma-delimited string containing one or more of the following values:

all

enables all types of auditing.

connect

enables auditing of both successful and failed connection attempts.

connectFailed

enables auditing of failed connection attempts.

DDL

enables auditing of DDL statements.

options

enables auditing of public options.

permission

enables auditing of permission checks, user checks, and SETUSER statements.

permissionDenied

enables auditing of failed permission and user checks.

triggers

enables auditing of a trigger event.

xp_cmdshell

enables auditing of xp_cmdshell invocations.

Remarks

Use sa_enable_auditing_type to specify which types of auditing to include. This system procedure adds the specified events to the current set of audit events. Use sa_disable_auditing_type to remove events from the current set of audit events. These system procedures set the PUBLIC *auditing_options* database option so the setting is permanent.

Set the PUBLIC *auditing* database option to On or Off to enable or disable auditing.

By default, all events are audited (types='all'). If you want a smaller set, use the sa_disable_auditing_type system procedure to clear the events you are not interested in; or use the sa_disable_auditing_type system procedure to clear all events and then use the sa_enable_auditing_type system procedure to specify which types of auditing you want.

If the set of events is empty and you set the PUBLIC *auditing* database option to On, no auditing information is recorded. To re-establish auditing, you must use the sa_enable_auditing_type system procedure to specify which types of information you want to audit.

If you set the PUBLIC *auditing* database option to Off, then no auditing information is recorded.

Specify the location where events are logged with the *audit_log* database option.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SET ANY SECURITY OPTION system privilege.

Side Effects

None

Example

The following example enables all auditing:

```
CALL sa_enable_auditing_type( 'all' );
```

The following example enables only DDL and triggers auditing:

```
CALL sa_disable_auditing_type( 'all' );
CALL sa_enable_auditing_type( 'DDL,triggers' );
```

The following example illustrates another way to enable only DDL and triggers auditing:

```
CALL sa_disable_auditing_type( 'all' );
CALL sa_enable_auditing_type( 'triggers' );
CALL sa_enable_auditing_type( 'DDL' );
```

Related Information

Database Activity Audits
sa_disable_auditing_type System Procedure [page 1582]
auditing Option

1.6.8.29 sa_eng_properties System Procedure

Reports database server property information.

Syntax

sa_eng_properties( )

Result Set

Column name	Data type	Description
PropNum	INTEGER	The database server property number.
PropName	VARCHAR(255)	The database server property name.
PropDescription	VARCHAR(255)	The database server property description.
Value	LONG VARCHAR	The database server property value.

Remarks

Returns the PropNum, PropName, PropDescription, and Value for each available server property. Values are returned for all database server properties and statistics related to database servers.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following statement returns a set of available server properties

```
CALL sa_eng_properties( );
```

PropNum	PropName	...
1	IdleWrite	...
2	IdleChkPt	...
...	...	...

Related Information

System Functions [page 224]
List of Database Server Properties
xp_cmdshell System Procedure [page 1864]

1.6.8.30 sa_error_stack_trace System Procedure

Returns the stack trace of the error that invoked the error handler.

Syntax

```
sa_error_stack_trace( )
```

Result Set

Column name	Data type	Description
StackLevel	UNSIGNED SMALLINT	The line number of the stack (1 for the top line). The statement that generated the error has the highest number.
UserName	CHAR(128)	The name of the owner of the procedure or NULL if the current level is in a batch.
ProcName	CHAR(128)	The name of the procedure where the statement was executed, or the batch type.
LineNumber	UNSIGNED INTEGER	The line number of the call within the procedure.
IsResignal	BIT	1 if the statement is a resignal, and 0 otherwise.

Remarks

Each row in the result set represents a single call on the call stack of the error. If the compound statement is not part of a procedure, function, trigger, or event, the type of batch (`<watcom_batch>` or `<tsql_batch>`) is returned instead of the procedure name.

This function returns line numbers as found in the proc_defn column of the SYSPROCEDURE system table for the procedure. These line numbers might differ from those of the source definition used to create the procedure.

This procedure returns the same information as the ERROR_STACK_TRACE function.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None.

Example

This example shows an example of calling the sa_error_stack_trace system procedure:

```
CALL sa_error_stack_trace();
```

This example shows the output of the sa_error_stack_trace system procedure with RESIGNAL:

```
CREATE OR REPLACE PROCEDURE error_reporting_procedure()
BEGIN
    SELECT *
    FROM sa_error_stack_trace();
END;
CREATE OR REPLACE PROCEDURE proc1()
BEGIN TRY
    BEGIN TRY
        DECLARE v INTEGER = 0;
        SET v = 1 / v;
    END TRY
    BEGIN CATCH
        CALL proc2();
    END CATCH
END TRY
BEGIN CATCH
    CALL error_reporting_procedure();
END CATCH;
CREATE OR REPLACE PROCEDURE proc2()
BEGIN
    CALL proc3();
END;
CREATE OR REPLACE PROCEDURE proc3()
BEGIN
    RESIGNAL;
END;
CALL proc1();
```

When the example above is run, the following result set is produced:

StackLevel	UserName	ProcName	LineNumber	IsResignal
1	DBA	proc1	8	0
2	DBA	proc2	3	0
3	DBA	proc3	3	1
4	DBA	proc1	5	0

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_stack_trace System Procedure [page 1725]

1.6.8.31 sa_event_schedules System Procedure

Displays schedule information about events.

Syntax

```
sa_event_schedules( evt_id )
```

Parameters

evt_id

This INTEGER parameter takes the ID number of an event.

Result Set

Column name	Data type	Description
sched_name	VARCHAR(128)	The name of the schedule the event runs under.
sched_def	LONG VARCHAR	The schedule definition.

Remarks

This procedure returns information about a specified event, including when the event is scheduled to run and how often it runs. If the event does not have a schedule, the procedure does not return a result set.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example retrieves the event schedule for the IncrementalBackup event.

```
call sa_event_schedules( (SELECT event_id FROM SYSEVENT WHERE
event_name='IncrementalBackup') );
```

sched_name	sched_def
IncrementalBackup	START TIME '01:00:00' EVERY 24 HOURS

Related Information

CREATE EVENT Statement [page 847]
EVENT_CONDITION Function [System] [page 369]
EVENT_CONDITION_NAME Function [System] [page 371]

EVENT_PARAMETER Function [System] [page 372]

1.6.8.32 sa_external_library_unload System Procedure

Unloads an external library.

Syntax

```
sa_external_library_unload( [ lib_name ] )
```

Parameters

lib_name

Use this optional LONG VARCHAR parameter to specify the name of a library to be unloaded. If no library is specified, all external libraries that are not in use are unloaded. The default is NULL.

Remarks

If an external library is specified, but is in use or is not loaded, an error is returned. If no parameter is specified, no error is returned.

The library name must match exactly the path and letter case of the original library specification. For example, the library name must match exactly the string following the '@' in the following EXTERNAL NAME clause.

```
EXTERNAL NAME 'xp_replicate@c:\\sqlany\\samples\\sqlanywhere\\ExternalProcedures\
\extproc.dll'
```

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY EXTERNAL OBJECT system privilege.

Side Effects

None

Example

The following example unloads an external library called `extproc.dll`:

```
CALL sa_external_library_unload( 'extproc.dll' );
```

The following example unloads all libraries that are not currently in use:

```
CALL sa_external_library_unload();
```

Related Information

External Call Interface

1.6.8.33 sa_flush_cache System Procedure

Empties all pages for the current database in the database server cache.

Syntax

```
sa_flush_cache( )
```

Remarks

Database administrators can use this procedure to empty the contents of the database server cache for the current database. This is useful in performance measurement to ensure repeatable results.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

The following example empties all pages for the current database in the database server cache.

```
CALL sa_flush_cache( );
```

1.6.8.34 sa_flush_statistics System Procedure

Saves all cost model statistics in the database server cache.

Syntax

sa_flush_statistics( )

Remarks

Use this procedure to flush current cost model statistics in the database, currently cached, to disk. You can then retrieve the statistics using the sa_get_histogram system procedure, or the Histogram utility (dbhist). When this system procedure runs, the ISYSCOLSTAT system table is updated. Under normal operation it should not be necessary to execute this procedure because the server automatically writes out statistics to disk on a periodic basis.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY STATISTICS or SERVER OPERATOR system privilege.

Side Effects

None

Example

The following example saves all cost model statistics in the database server cache.

```
CALL sa_flush_statistics( );
```

Related Information

sa_get_histogram System Procedure [page 1601]
SYSCOLSTAT System View [page 1902]
Histogram Utility (dbhist)

1.6.8.35 sa_get_bits System Procedure

Takes a bit string and returns a row for each bit in the string. By default, only rows with a bit value of 1 are returned.

Syntax

```
sa_get_bits(
bit_string
[, only_on_bits ]
)
```

Parameters

bit_string

Use this LONG VARBIT parameter to specify the bit string from which to get the bits. If the `bit_string` parameter is NULL, no rows are returned.

only_on_bits

Use this optional BIT parameter to specify whether to return only rows with on bits (bits with the value of 1). Specify 1 (the default) to return only rows with on bits; specify 0 to return rows for all bits in the bit string.

Result Set

Column	Data type	Description
bitnum	UNSIGNED INTEGER	The position of the bit described by this row. For example, the first bit in the bit string has bitnum of 1.
bit_val	BIT	The value of the bit at position bitnum. If `only_on_bits` is set to 1, this value is always 1.

Remarks

The sa_get_bits system procedure decodes a bit string, returning one row for each bit in the bit string, indicating the value of the bit. If `only_on_bits` is set to 1 (the default) or NULL, then only rows corresponding to on bits are returned. An optimization allows this case to be processed efficiently for long bit strings that have few on bits. If `only_on_bits` is set to 0, then a row is returned for each bit in the bit string.

For example, the statement `CALL sa_get_bits( '1010' )` returns the following result set, indicating on bits in positions 1 and 3 of the bit string.

bitnum	bit_val
1	1
3	1

The sa_get_bits system procedure can be used to convert a bit string into a relation. This can be used to join a bit string with a table, or to retrieve a bit string as a result set instead of as a single binary value. It can be more efficient to retrieve a bit string as a result set if there are a large number of 0 bits, as these do not need to be retrieved.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example shows how to use the sa_get_bits system procedure to encode a set of integers as a bit string, and then decode it for use in a join:

```
CREATE VARIABLE @s_depts LONG VARBIT;
SELECT  SET_BITS( DepartmentID )
 INTO @s_depts
 FROM Departments
 WHERE DepartmentName like 'S%';
SELECT *
 FROM sa_get_bits( @s_depts ) B
  JOIN Departments D ON B.bitnum = D.DepartmentID;
```

Related Information

sa_split_list System Procedure [page 1722]
SET_BIT Function [Bit Array] [page 548]
SET_BITS Function [Aggregate] [page 549]
GET_BIT Function [Bit Array] [page 389]

1.6.8.36 sa_get_dtt System Procedure

Reports the current value of the Disk Transfer Time (DTT) model, which is part of the cost model.

Syntax

```
sa_get_dtt( file_id )
```

Parameters

file_id

Use this UNSIGNED SMALLINT parameter to specify the database file ID.

Result Set

Column name	Data type	Description
BandSize	UNSIGNED INTEGER	Size, in pages, of disk over which random access takes place.
ReadTime	UNSIGNED INTEGER	Amortized cost, in microseconds, of reading one page.
WriteTime	UNSIGNED INTEGER	Amortized cost, in microseconds, of writing one page.
QueueDepth	UNSIGNED INTEGER	The number of outstanding I/Os in the disk's I/O queue over which random access takes place.

Remarks

You can obtain the `file_id` from the SYSDBSPACE system view.

This procedure retrieves data from the ISYSOPTSTAT system table.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example reports the current value of the Disk Transfer Time (DTT) model for the system dbspace.

```
CALL sa_get_dtt( (select dbspace_id from SYSDBSPACE where
dbspace_name='system') );
```

Related Information

SYSDBSPACE System View [page 1904]
SYSOPTSTAT System View [page 1930]
sa_get_dtt_groupreads System Procedure [page 1599]

1.6.8.37 sa_get_dtt_groupreads System Procedure

Estimates and reports the cost of issuing group reads on the database server.

Syntax

```
sa_get_dtt_groupreads( dbspace_id )
```

Parameters

dbspace_id

Use this UNSIGNED SMALLINT parameter to specify the database file ID.

Result Set

Column name	Data type	Description
GroupSize	UNSIGNED INTEGER	Size, in pages, of disk over which random access takes place.
ReadTime	FLOAT	Amortized cost, in microseconds, of reading one page.

Remarks

You can obtain the `dbspace_id` from the SYSDBSPACE system view. The estimates returned by the sa_get_dtt_groupreads system procedure are part of the cost model, and are used to select group reads of appropriate sizes during operations such as sorting.

This procedure, intended for internal diagnostic purposes, retrieves data from the ISYSOPTSTAT system table. If no entries are recorded in this table, typical values are returned. To tailor estimates for your hardware, execute the following statement:

```
ALTER DATABASE CALIBRATE GROUP READ;
```

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example reports the cost of issuing group reads for the system dbspace.

```
CALL sa_get_dtt_groupreads( (select dbspace_id from SYSDBSPACE where
dbspace_name='system') );
```

Related Information

SYSDBSPACE System View [page 1904]
SYSOPTSTAT System View [page 1930]
ALTER DATABASE Statement [page 662]
sa_get_dtt System Procedure [page 1598]

1.6.8.38 sa_get_histogram System Procedure

Retrieves the histogram for a column.

Syntax

```
sa_get_histogram(
col_name
, tbl_name
[, owner_name ]
)
```

Parameters

col_name

Use this CHAR(128) parameter to specify the column for which to retrieve the histogram.

tbl_name

Use this CHAR(128) parameter to specify the table in which `col_name` is found.

owner_name

Use this optional CHAR(128) parameter to specify the owner of `tbl_name`. The default is NULL.

Result Set

Column name	Data type	Description
StepNumber	SMALLINT	Histogram bucket number. The frequency of the first bucket (StepNumber = 0) indicates the selectivity of NULLs.
Low	CHAR(128)	Lowest (inclusive) column value in the bucket.

Column name	Data type	Description
High	CHAR(128)	Highest (exclusive) column value in the bucket.
Frequency	DOUBLE	Selectivity of values in the bucket.

Remarks

This procedure, intended for internal diagnostic purposes, retrieves column statistics from the database server for the specified columns. While these statistics are permanently stored in the ISYSCOLSTAT system table, they are maintained in memory while the server is running, and written to ISYSCOLSTAT periodically. As such, the statistics returned by the sa_get_histogram system procedure may differ from those obtained by selecting from ISYSCOLSTAT at any given point of time.

You can manually update ISYSCOLSTAT with the latest statistics held in memory using the sa_flush_statistics system procedure, however, this is not recommended in a production environment, and should be reserved for diagnostic purposes.

A singleton bucket is indicated by a Low value in the result set being equal to the corresponding High value.

Use the Histogram utility to view histograms.

To determine the selectivity of a predicate over a string column, use the ESTIMATE or ESTIMATE_SOURCE functions. For string columns, both sa_get_histogram and the Histogram utility retrieve nothing from the ISYSCOLSTAT system table. Attempting to retrieve string data generates an error.

Statistics (including histograms) may not be present for a table or materialized view, for example, if statistics were recently dropped. In this case, the result set for the sa_get_histogram system procedure is empty. To create statistics for a table or materialized view, execute a CREATE STATISTICS statement.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY STATISTICS system privilege.

Side Effects

None

Example

For example, the following statement retrieves the histogram for the ProductID column of the SalesOrderItems table:

```
CALL sa_get_histogram( 'ProductID', 'SalesOrderItems' );
```

Related Information

Optimizer Estimates and Statistics
Histogram Utility (dbhist)
SYSCOLSTAT System View [page 1902]
CREATE STATISTICS Statement [page 991]
sa_flush_statistics System Procedure [page 1595]
ESTIMATE_SOURCE Function [Miscellaneous] [page 367]
ESTIMATE Function [Miscellaneous] [page 366]
Histogram Utility (dbhist)

1.6.8.39 sa_get_ldapserver_status System Procedure

Allows you to determine the current status of LDAP servers.

Syntax

sa_get_ldapserver_status( )

Result Set

Column name	Data type	Description
ldsrv_id	UNSIGNED BIGINT	A unique number identifying the LDAP server.
ldsrv_name	CHAR(128)	The name of the LDAP server.
ldsrv_state	CHAR(9)	The current state of the LDAP server at last checkpoint.

Column name	Data type	Description
ldsrv_last_state_change	TIMESTAMP	The date and time that the state was changed. Regardless of the database server local time zone, the value is stored in Coordinated Universal Time (UTC). This value is not affected by simulated time zone.

Remarks

This procedure returns a result set that shows the current status of LDAP servers.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example uses the sa_get_ldapserver_status system procedure to return the status of all LDAP servers.

```
CALL sa_get_ldapserver_status;
```

1.6.8.40 sa_get_request_profile System Procedure

Analyzes the request log to determine the execution times of similar statements.

Syntax

```
sa_get_request_profile(
   [ filename
   [, conn_id
   [, first_file
   [, num_files ] ] ] ]
```

```
)
```

Parameters

filename

Use this optional LONG VARCHAR parameter to specify the request logging file name. The default is NULL.

conn_id

Use this optional UNSIGNED INTEGER parameter to specify the connection ID number. The default is 0.

first_file

Use this optional INTEGER parameter to specify the first request log file to analyze. The default is -1.

num_files

Use this optional INTEGER parameter to specify the number of request log files to analyze. The default is 1.

Remarks

This procedure calls sa_get_request_times to process a request log file, and then summarizes the results into the global temporary table SATMP_request_profile. This table contains the statements from the log along with how many times each was executed, and their total, average, and maximum execution times. The table can be sorted in various ways to identify targets for performance optimization efforts.

If you do not specify a request log file (`filename`), the default is the current log file that is specified with the -zo database server option, or that has been specified by

```
sa_server_option( 'RequestLogFile', filename )
```

If a connection ID is specified, it is used to filter information from the log so that only requests for that connection are retrieved.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the DIAGNOSTICS system role, and the MANAGE PROFILING system privilege.

Side Effects

Automatic commit

Example

The following command obtains the request times for the requests in the file `req.out`.

```
CALL sa_get_request_profile('req.out');
```

The following command obtains the request times for the requests in the files `req.out.3`, `req.out.4`, and `req.out.5`.

```
CALL sa_get_request_profile('req.out',0,3,3);
```

Related Information

sa_get_request_times System Procedure [page 1606]
sa_statement_text System Procedure [page 1728]
sa_server_option System Procedure [page 1698]
-zo Database Server Option

1.6.8.41 sa_get_request_times System Procedure

Analyzes the request log to determine statement execution times.

Syntax

```
sa_get_request_times( [ filename
   [, conn_id
   [, first_file
   [, num_files ] ] ] ]
)
```

Parameters

filename

Use this optional LONG VARCHAR parameter to specify the request logging file name. The default is NULL.

conn_id

Use this optional UNSIGNED INTEGER parameter to specify the connection ID number. The default is 0.

first_file

Use this optional INTEGER parameter to specify the first file to analyze. The default is -1.

num_files

Use this optional INTEGER parameter to specify the number of request log files to analyze. The default is 1.

Remarks

This procedure reads the specified request log and populates the global temporary table SATMP_request_time with the statements from the log and their execution times.

For statements such as inserts and updates, the execution time is straightforward. For queries, the time is calculated from preparing the statement to dropping it, including describing it, opening a cursor, fetching rows, and closing the cursor. For most queries, this is an accurate reflection of the amount of time taken. When the cursor is left open while other events take place, such as operator interaction or client processing, the time appears as a large value but is not a true indication that the query is costly.

This procedure recognizes host variables in the request log and populates the global temporary table SATMP_request_hostvar with their values. For older databases where this temporary table does not exist, host variable values are ignored.

If you do not specify a request log file, the default is the current log file that is specified in the command with -zo, or that has been specified by:

```
call sa_server_option( 'RequestLogFile', filename )
```

If a connection ID is specified, it is used to filter information from the log so that only requests for that connection are retrieved.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE PROFILING or MONITOR system privilege.

Side Effects

Automatic commit

Example

The following command obtains the execution times for the requests in the file `req.out`.

```
CALL sa_get_request_times('req.out');
```

The following command obtains the execution times for the requests in the files `req.out.3`, `req.out.4`, and `req.out.5`.

```
CALL sa_get_request_times('req.out',0,3,3);
```

Related Information

sa_get_request_profile System Procedure [page 1604]
sa_statement_text System Procedure [page 1728]
sa_server_option System Procedure [page 1698]

1.6.8.42 sa_get_server_messages System Procedure (Deprecated)

Allows you to return constants from the database server messages window as a result set. This system procedure is deprecated. Use the sa_server_messages system procedure instead.

Syntax

```
sa_get_server_messages( first_line )
```

Parameters

first_line

Use this INTEGER parameter to specify the line number from which to start displaying server messages.

Result Set

Column name	Data type	Description
line_num	INTEGER	The line number of a server message.
message_text	VARCHAR(255)	The server message text.
message_time	TIMESTAMP	The time of the message.

Remarks

This procedure takes an INTEGER parameter that specifies the starting line number to display, and returns a row for that line and for all subsequent lines. If the starting line is negative, the result set starts at the first available line. The result set includes the line number, message text, and message time.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example uses the sa_get_server_messages system procedure to return a result set containing the content of the database server messages window, starting from line 16.

```
CALL sa_get_server_messages( 16 );
```

line_num	message_text	...
16	Running Windows 7 Build 7601...	...
17	Server built for X86_64 processor architecture	...
...	...	...

1.6.8.43 sa_get_table_definition System Procedure

Returns a LONG VARCHAR string containing the SQL statements required to create the specified table and its indexes, foreign keys, triggers, and granted privileges.

Syntax

```
sa_get_table_definition(
@owner
, @tabname
)
```

Parameters

@owner

Use this VARCHAR(128) parameter to specify the owner of `tabname`.

@tabname

Use this VARCHAR(128) parameter to specify the name of the table.

Remarks

This function returns a LONG VARCHAR string containing the SQL statements required to create the specified table and its indexes, foreign keys, triggers, and granted privileges. To create a new table with the same definition, use the string returned by this function with the EXECUTE IMMEDIATE statement and the LOCATE, SUBSTRING, and REPLACE functions.

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must have the SELECT ANY TABLE system privilege, or SELECT privilege on the SYSUSERPERM compatibility view.

Side Effects

None

Example

The following statement uses the sa_get_table_definition function to display the string containing the SQL statements required to create the Departments table.

```
SELECT sa_get_table_definition( 'GROUPO', 'Departments' );
```

Related Information

SYSUSERPERM Compatibility View (Deprecated) [page 2009]
sa_split_list System Procedure [page 1722]
EXECUTE IMMEDIATE Statement [SP] [page 1155]
LOCATE Function [String] [page 442]
SUBSTRING Function [String] [page 578]
REPLACE Function [String] [page 525]

1.6.8.44 sa_get_user_status System Procedure

Allows you to determine the current status of users.

Syntax

```
sa_get_user_status( )
```

Result Set

Column name	Data type	Description
user_id	UNSIGNED INTEGER	A unique number identifying the user.
user_name	CHAR(128)	The name of the user.
connections	INTEGER	The current number of connections by this user.
failed_logins	UNSIGNED INTEGER	The number of failed login attempts made by the user.
last_login_time	TIMESTAMP	Returns the database server local date and time (accurate to hours and minutes) that the most recent connection was established. This value is affected by simulated time zone.
locked	TINYINT	Indicates if the user account is locked.
reason_locked	LONG VARCHAR	The reason the account is locked.
user_dn	CHAR(1024)	The Distinguished Name (DN) for a user ID connecting to an LDAP server.
user_dn_cached_at	TIMESTAMP	The date and time that the user_dn column was last cached. This value is used to determine whether to purge an old DN. Regardless of the database server local time zone, the value is stored in Coordinated Universal Time (UTC). This value is not affected by simulated time zone.
password_change_state	BIT	A value that indicates whether a dual password change is in progress (0=No, 1=Yes). The default is 0.
password_change_first_user	UNSIGNED INTEGER	The user_id of the user who set the first part of a dual password; otherwise NULL.
password_change_second_user	UNSIGNED INTEGER	The user_id of the user who set the second part of a dual password; otherwise NULL.

Remarks

This procedure returns a result set that shows the current status of users. In addition to basic user information, the procedure includes a column indicating if the user has been locked out and a column with a reason for the lockout. Users can be locked out for the following reasons: locked due to policy, password expiry, or too many failed attempts.

Privileges

You must have EXECUTE privilege on the system procedure.

To view information about other users, you must also have the MANAGE ANY USER system privilege.

Side Effects

None

Example

The following example uses the sa_get_user_status system procedure to return the status of database users.

```
CALL sa_get_user_status;
```

Related Information

Login Policies
Dual Control Passwords
Creating a Login Policy
Creating a User (SQL Central)
Assigning a Login Policy to an Existing User
Altering a Login Policy
Deleting a Login Policy

1.6.8.45 sa_http_header_info System Procedure

Returns HTTP request header names and values.

Syntax

```
sa_http_header_info( [header_parm] )
```

Parameters

header_parm

Use this optional VARCHAR(255) parameter to specify an HTTP header name. The default is NULL.

Result Set

Column name	Data type	Description
Name	VARCHAR(255)	The HTTP header name.
Value	LONG VARCHAR	The HTTP header value.

Remarks

The sa_http_header_info system procedure returns header names and values. If you do not specify the header name using the optional parameter, the result set contains values for all headers.

This procedure returns a non-empty result set if it is called while processing an HTTP request within a web service.

Note

The sa_http_header_info system procedure may return multiple rows with the same name if the request contains multiple HTTP headers with the same name.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following web service procedure which is called from a web service illustrates the use of the sa_http_header_info system procedure.

```
CREATE OR REPLACE PROCEDURE User1.HTTPHeaderExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
    DECLARE myname VARCHAR(255);
    DECLARE myvalue LONG VARCHAR;
    DECLARE err_notfound
        EXCEPTION FOR SQLSTATE '02000';
    DECLARE curs CURSOR FOR
        SELECT Name, Value FROM sa_http_header_info();
    MESSAGE '=== HTTP Headers ===' TO CONSOLE;
    OPEN curs;
    FetchLoop: LOOP
        FETCH next curs INTO myname, myvalue;
        IF SQLSTATE = err_notfound THEN
            LEAVE FetchLoop;
        END IF;
        MESSAGE myname, '=', myvalue TO CONSOLE;
    END LOOP FetchLoop;
    CLOSE curs;
END;
```

When the web service that calls this web service procedure is used, output appears in the database server messages window that is similar to the following.

```
=== HTTP Headers ===
@HttpQueryString=param1=value1&param2=value2&param3=value3
User-Agent=Mozilla/5.0 (Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101 Firefox/
16.0
Authorization=Basic VXNlcjE6dXNlcg==
Cache-Control=max-age=0
Connection=keep-alive
Host=webserver-t3500.sap.com:8082
@HttpURI=/ShowHTTPHeaders?param1=value1&param2=value2&param3=value3
@HttpMethod=GET
Accept=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
@HttpVersion=HTTP/1.1
Accept-Language=en-US,en;q=0.5
Accept-Encoding=gzip, deflate
```

Related Information

Web Services Functions [page 221]
NEXT_HTTP_HEADER Function [Web Service] [page 472]
HTTP_HEADER Function [Web Service] [page 412]

Web Services System Procedures [page 1512]

1.6.8.46 sa_http_php_page System Procedure

Returns the result of passing the PHP code that is to be interpreted through a PHP interpreter using the current HTTP request for context information such as headers, GET/POST data, protocol version, request URL, method, and so on.

Syntax

```
sa_http_php_page( php_page )
```

Parameters

php_page

This LONG VARCHAR parameter contains the entire PHP code that is to be interpreted, including the starting and ending markers (`<?php` and `?>`).

Returns

This function returns a LONG BINARY value.

Remarks

To use this system procedure, the PHP external environment must already be installed.

The owner of this system procedure is DBO. However, for improved security, the sa_http_php_page system procedure is executed as the invoker.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example submits the phpinfo() query to a PHP interpreter and displays the HTML result.

```
SELECT CAST( sa_http_php_page('<?php phpinfo(); ?>') AS LONG VARCHAR );
```

The following example submits a PHP script to a PHP interpreter and displays the XML result.

```
SELECT CAST( sa_http_php_page('<?php '||
    '$conn = sasql_connect( "UID=DBA;PWD=sql" ); '||
    '$result = sasql_query( $conn, "SELECT * FROM Employees" ); '||
    'sasql_result_all( $result ); '||
    'sasql_free_result( $result ); '||
    'sasql_disconnect( $conn ); '||
    '?>') AS LONG VARCHAR );
```

Related Information

Web Services Functions [page 221]
sa_http_php_page_interpreted System Procedure [page 1616]
Web Services System Procedures [page 1512]

1.6.8.47 sa_http_php_page_interpreted System Procedure

Returns the result of passing the PHP code that is to be interpreted through a PHP interpreter using the specified parameters for context information such as headers, GET/POST data, protocol version, request URL, method, and so on.

Syntax

```
sa_http_php_page_interpreted(
php_page
, method
, url
, version
, headers
, request_body
)
```

Parameters

php_page

This LONG VARCHAR parameter contains the entire PHP code that is to be interpreted, including the starting and ending markers (`<?php` and `?>`).

method

This LONG VARCHAR parameter contains the HTTP request method (for example, GET, POST, PUT, or one of the other standard request methods). The value for `method` can be determined using the value of @HttpMethod in the current HTTP request.

url

This LONG VARCHAR parameter contains the full HTTP request URL, including the query string, if present. The value for `url` can be determined using the value of @HttpURI in the current HTTP request.

version

This LONG VARCHAR parameter contains the HTTP request protocol version (for example, HTTP/1.1). The value for `version` can be determined using the value of @HttpVersion in the current HTTP request.

headers

This LONG BINARY parameter contains the HTTP request headers in the standard HTTP header format: `Field-Name: Value\r\n`. The value for headers can be retrieved from the current HTTP request using the following SELECT statement:

```
SELECT LIST( name || ': ' || value, CHAR(13) || CHAR(10) )
   FROM sa_http_header_info();
```

request_body

This LONG BINARY parameter contains the HTTP request body in binary form. The value for `request_body` can be retrieved from the current HTTP request using the HTTP_BODY function.

Returns

This function returns a LONG BINARY value.

Remarks

To use this system procedure, the PHP external environment must already be installed.

To use this system procedure outside web services requests, you must provide request information. Any headers set within the PHP code are lost.

The owner of this system procedure is DBO. However, for improved security, the sa_http_php_page_interpreted system procedure is executed as the invoker.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example submits the phpinfo() query to a PHP interpreter and displays the HTML result.

```
BEGIN
    DECLARE headers LONG VARCHAR;
    SELECT list( name || ': ' || value, char(13) || char(10) ) INTO headers
    FROM sa_http_header_info();
    SELECT CAST( sa_http_php_page_interpreted( '<?php phpinfo(); ?>',
        http_header( '@HttpMethod' ),
        http_header( '@HttpURI' ),
        http_header( '@HttpVersion' ),
        headers,
        HTTP_BODY() ) AS LONG VARCHAR);
END;
```

Related Information

The PHP External Environment
Web Services Functions [page 221]
HTTP_BODY Function [Web Service] [page 408]
sa_http_php_page System Procedure [page 1615]
sa_http_header_info System Procedure [page 1613]
Web Services System Procedures [page 1512]

1.6.8.48 sa_http_variable_info System Procedure

Returns HTTP variable names and values.

Syntax

```
sa_http_variable_info( [variable_parm] )
```

Parameters

variable_parm

Use this optional VARCHAR(255) parameter to specify an HTTP variable name. The default is NULL.

Result Set

Column name	Data type	Description
Name	VARCHAR(255)	The HTTP variable name.
Value	LONG VARCHAR	The HTTP variable value.

Remarks

The sa_http_variable_info system procedure returns variable names and values. If you do not specify the variable name using the optional parameter, the result set contains values for all variables.

This procedure returns a non-empty result set if it is called while processing an HTTP request within a web service.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following web service procedure which is called from a web service illustrates the use of the sa_http_variable_info system procedure.

```
CREATE OR REPLACE PROCEDURE User1.HTTPVariableExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
    DECLARE myname VARCHAR(255);
    DECLARE myvalue LONG VARCHAR;
```

```
    DECLARE err_notfound
        EXCEPTION FOR SQLSTATE '02000';
    DECLARE curs CURSOR FOR
        SELECT Name, Value FROM sa_http_variable_info();
    MESSAGE '=== HTTP Variables ===' TO CONSOLE;
    OPEN curs;
    FetchLoop: LOOP
        FETCH next curs INTO myname, myvalue;
        IF SQLSTATE = err_notfound THEN
            LEAVE FetchLoop;
        END IF;
        MESSAGE myname, '=', myvalue TO CONSOLE;
    END LOOP FetchLoop;
    CLOSE curs;
END;
```

For a URL parameter list like `?param1=value1&param2=value2&param3=value3`, output from this sample web service procedure appears in the database server messages window as a list in the form of parameter name=value.

```
=== HTTP Variables ===
param1=value1
param3=value3
param2=value2
```

Related Information

Web Services Functions [page 221]
NEXT_HTTP_VARIABLE Function [Web Service] [page 475]
HTTP_VARIABLE Function [Web Service] [page 417]
sa_http_header_info System Procedure [page 1613]
Web Services System Procedures [page 1512]

1.6.8.49 sa_index_density System Procedure

Reports information about the amount of fragmentation and skew within indexes.

Syntax

```
sa_index_density(
[ tbl_name
[, owner_name ] ]
)
```

Parameters

tbl_name

Use this optional CHAR(128) parameter to specify the table name. The default is NULL.

owner_name

Use this optional CHAR(128) parameter to specify the owner name. The default is NULL.

Result Set

Column name	Data type	Description
TableName	CHAR(128)	The name of a table.
TableId	UNSIGNED INTEGER	The table ID.
IndexName	CHAR(128)	The name of an index.
IndexId	UNSIGNED INTEGER	The index ID. This column contains one of the following values: **0** for primary keys **SYSFKEY.foreign_key_id** for foreign keys **SYSIDX.index_id** for all other indexes
IndexType	CHAR(4)	The index type. This column contains one of the following values: **PKEY** for primary keys **FKEY** for foreign keys **UI** for unique indexes **UC** for unique constraints **NUI** for non-unique indexes
LeafPages	UNSIGNED INTEGER	The number of leaf pages.
Density	DOUBLE	A fraction between 0 and 1 that provides an indication of how full each index page is (on average).

Column name	Data type	Description
Skew	DOUBLE	A number that provides an indication of the level of unbalance in an index. A value of 1 indicates a perfectly balanced index. Larger values indicate a higher degree of skew.

Remarks

Use the sa_index_density system procedure to obtain information about the degree of fragmentation and skew in indexes. For indexes with a high number of leaf pages, higher density values and lower skew values are desirable.

Index density reflects the average fullness of the index pages, as a percentage. A density of 0.7 indicates that index pages are, on average, 70% full with index data. Index skew reflects the typical deviation from the average density. The amount of skew is important to the optimizer when making selectivity estimates.

When the number of leaf pages is low, you do not need to be concerned about density and skew values. Density and skew values become important only when the number of leaf pages are high. When the number of leaf pages is high, a low density value can indicate fragmentation, and a high skew value can indicate that indexes are not well balanced. Both of these can be factors in poor performance. Executing a REORGANIZE TABLE statement addresses both of these issues.

If you do not specify a table when calling this procedure, the information for all indexes on all tables in the database is returned.

Privileges

You must have EXECUTE privilege on the system procedure, as well as one of the following system privileges:

- MONITOR
- MANAGE ANY STATISTICS
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX
- CREATE ANY OBJECT
- ALTER ANY OBJECT
- DROP ANY OBJECT

Side Effects

None

Example

The following example uses the sa_index_density system procedure to return a result set summarizing the amount of fragmentation and skew within all the indexes in the database.

```
CALL sa_index_density( );
```

Related Information

Running a Comprehensive Profiling Session (Profiler)
REORGANIZE TABLE Statement [page 1336]

1.6.8.50 sa_index_levels System Procedure

Assists in performance tuning by reporting the number of levels in an index.

Syntax

```
sa_index_levels(
[ tbl_name
[, owner_name ] ]
)
```

Parameters

tbl_name

Use this optional CHAR(128) parameter to specify the table name. The default is NULL.

owner_name

Use this optional CHAR(128) parameter to specify the owner name. The default is NULL.

Result Set

Column name	Data type	Description
TableName	CHAR(128)	The name of a table.
TableId	UNSIGNED INTEGER	The table ID.
IndexName	CHAR(128)	The name of an index.

Column name	Data type	Description
IndexId	UNSIGNED INTEGER	The index ID. This column contains one of the following: **0** for primary keys **SYSFKEY.foreign_key_id** for foreign keys **SYSIDX.index_id** for all other indexes
IndexType	CHAR(4)	The index type. This column contains one of the following values: **PKEY** for primary keys **FKEY** for foreign keys **UI** for unique indexes **UC** for unique constraints **NUI** for non-unique indexes
Levels	INTEGER	The number of levels in the index.

Remarks

The number of levels in the index tree determines the number of I/O operations needed to access a row using the index. Indexes with a few levels are more efficient than indexes with a large number of levels.

The procedure returns a result set containing the table name, the table ID, the index name, the index ID, the index type, and the number of levels in the index.

If no arguments are supplied, levels are returned for all indexes in the database. If only `tbl_name` is supplied, levels for all indexes on that table are supplied. If `tbl_name` is NULL and an `owner_name` is given, only levels for indexes on tables owned by that user are returned.

Privileges

You must have EXECUTE privilege on the system procedure, as well as one of the following system privileges:

- MANAGE ANY STATISTICS
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX
- CREATE ANY OBJECT
- ALTER ANY OBJECT
- DROP ANY OBJECT

Side Effects

None

Example

The following example uses the sa_index_levels system procedure to return the number of levels in the Products index.

```
CALL sa_index_levels( );
```

TableName	TableId	IndexName	...	Levels
Products	436	Products	...	1
...	...	...	...	...

Related Information

CREATE INDEX Statement [page 886]

1.6.8.51 sa_install_feature System Procedure

Installs additional features, for example additional spatial features.

Syntax

```
sa_install_feature( feat_name )
```

Parameters

feat_name

A LONG VARCHAR parameter that identifies the feature to install. The default is NULL. The supported feature names are:

Value	Description
st_geometry_predefined_uom	Installs predefined units of measure that are not installed by default in new databases.
st_geometry_predefined_srs	Installs predefined spatial reference systems and units of measure that are not installed by default in new databases.
st_geometry_compat_func	Installs a set of spatial compatibility functions. These functions can be used as an alternative to the spatial methods.

Feature name definitions are provided in the `st_geometry_config.tgz` file located in the *%SQLANY17%* `\scripts` directory. If the file is removed and you attempt to install features that are dependent on the file, an error is returned.

Remarks

You can query the feat_name value to see what will be installed. For example, the following query returns the units of measure that would be installed for st_geometry_predefined_uom.

```
SELECT * FROM st_geometry_predefined_uom( 'CREATE' );
```

The previous example also shows you parameter names so you can query for specific values using a WHERE clause. For example, the following statement queries the unit_name parameter for the chain unit of measure:

```
SELECT * FROM st_geometry_predefined_uom( 'CREATE' ) WHERE unit_name='chain';
```

unit_name	unit_type	conversion_factor	...
chain	LINEAR	20.1168	...

The following returns all units of measure that are based on foot:

```
SELECT * FROM st_geometry_predefined_uom() WHERE unit_name LIKE '%foot%';
```

Use the following query to find the spatial reference systems that would be installed:

```
SELECT * FROM st_geometry_predefined_srs();
```

The following statement queries for a spatial reference system by **`organization`** and **`organization_coordsys_id`**:

```
SELECT * FROM st_geometry_predefined_srs() WHERE organization='EPSG' AND
organization_coordsys_id=2295;
```

Privileges

You must have EXECUTE privilege on the system procedure.

For st_geometry_predefined_uom and st_geometry_predefined_srs, you must also have the MANAGE ANY SPATIAL OBJECT system privilege.

For st_geometry_compat_func, you must also have the MANAGE ANY OBJECT PRIVILEGE, CREATE ANY PROCEDURE, and SELECT ANY TABLE system privileges.

For sa_install_feature, you must also have the MANAGE ANY SPATIAL OBJECT system privilege.

Example

The following statement installs all of the predefined units of measure that are not installed by default in a new database:

```
CALL sa_install_feature( 'st_geometry_predefined_uom' );
```

The following statement installs a set of spatial compatibility functions which can be used as an alternative to the spatial methods:

```
CALL sa_install_feature( 'st_geometry_compat_func' );
```

Related Information

Units of Measure
Spatial Compatibility Functions

1.6.8.52 sa_java_loaded_classes System Procedure

Lists the classes currently loaded by the database server into a Java VM.

Syntax

```
sa_java_loaded_classes( )
```

Result Set

Column name	Data type	Description
class_name	VARCHAR(512)	The name of a class currently loaded by the database server into a Java VM.

Remarks

Returns a result set containing all the names of the Java classes currently loaded by the database server into a Java VM.

The procedure can be useful to diagnose missing classes. It can also be used to identify which classes from a particular jar are used by a given application.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example calls the init cover function to load and call the init method of the sample Invoice Java class. It then calls the sa_java_loaded_classes to obtain a list of all loaded Java classes.

```
CALL init( 'Work boots', 79.99, 'Hay fork', 37.49 );
CALL sa_java_loaded_classes( );
```

Related Information

How to Install Java Classes into a Database
Tutorial: Using Java in the Database

1.6.8.53 sa_list_cursors System Procedure

Returns the list of cursors in use by the current connection.

Syntax

```
sa_list_cursors( )
```

Result Set

Column name	Data type	Description
handle	UNSIGNED INTEGER	A unique handle identifying the cursor.
scope	INTEGER	The scope of the call stack where the cursor is open.
cursor_name	VARCHAR(128)	The cursor name.
is_open	BIT	The indicator of whether the cursor is currently open (1).
is_pinned	BIT	The indicator of whether the cursor is currently pinned in memory (1) in anticipation of reuse.
fetch_count	UNSIGNED BIGINT	The number of rows that have been fetched from the cursor.

Remarks

The sa_list_cursors system procedure can be used in a CALL statement or in the FROM clause of a SELECT statement.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns the list of open cursors for the connection:

```
CALL sa_list_cursors();
```

Related Information

sa_copy_cursor_to_temp_table System Procedure [page 1559]
sa_describe_cursor System Procedure [page 1573]

1.6.8.54 sa_list_statements System Procedure

Returns the list of statements in use by the current connection.

Syntax

```
sa_list_statements( )
```

Result Set

Column name	Data type	Description
handle	UNSIGNED INTEGER	A unique handle identifying the statement.
statement_number	INTEGER	The number of the statement within the connection.
num_client_prepares	UNSIGNED INTEGER	The number of times the client has prepared the identical statement text.
num_opens	UNSIGNED INTEGER	The number of times the statement has been opened.
schema_version_on_create	UNSIGNED INTEGER	For internal use only.
dropped_by_app	BIT	If the statement has been dropped by the client but retained for caching the value is 1; otherwise, the value is 0.
invalid_cached_statement	BIT	If this is a cached statement that is no longer valid (for example, due to schema changes) the value is 1; otherwise the value is 0.

Column name	Data type	Description
SQLStatement	LONG VARCHAR	The text of the statement.

Remarks

The sa_list_statements system procedure can be used in a CALL statement or in the FROM clause of a SELECT statement. The statement executing the sa_list_statements system procedure is not included in the result.

The SQLStatement column is rewritten from the original text due to semantic transform optimizations and normalization in a manner similar to that of the REWRITE function. Sensitive information such as encryption keys and passwords is replaced with ***. Because of these changes, the SQLStatement value requires interpretation when comparing to statements in the application, which might have a different form.

Privileges

None

Side Effects

None

Example

The following example returns the list of statements for the connection:

```
CALL sa_list_statements();
```

The following example returns the list of statements that contribute to the max_statement_count resource governor:

```
SELECT *
FROM sa_list_statements()
WHERE dropped_by_app=0;
```

1.6.8.55 sa_load_cost_model System Procedure

Replaces the current cost model with the cost model stored in the specified file.

Syntax

```
sa_load_cost_model( file_name )
```

Parameters

file_name

Use this CHAR(1024) parameter to specify the name of the cost model file to load.

Remarks

The optimizer uses cost models to determine optimal access plans for queries. The database server maintains a cost model for each database. The cost model for a database can be recalibrated at any time using the CALIBRATE SERVER clause of the ALTER DATABASE statement. For example, you might decide to recalibrate the cost model if you move the database onto non-standard hardware.

The sa_load_cost_model system procedure allows you to load a cost model that has been saved to file (`file_name`). Loading a cost model replaces the current cost model for the database.

Note

The sa_unload_cost_model system procedure does not include CALIBRATE PARALLEL READ information in the file that sa_load_cost_model loads.

Using the sa_load_cost_model system procedure can eliminate repetitive, time-consuming recalibration activities when there is a large number of identical hardware installations.

Exclusive use of the database is required when loading the new cost model.

When loading a cost model, consider whether it was generated for a database that is located on similar hardware. Loading a cost model from a database that is stored on significantly different hardware may cause poor performance due to inefficient access plans.

Cost models are saved to file using the sa_unload_cost_model system procedure.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the ALTER DATABASE and LOAD ANY TABLE system privileges.

Side Effects

The database server performs a COMMIT after loading the new cost model.

Example

The following example loads the cost model from a file called costmodel8:

```
CALL sa_load_cost_model( 'costmodel8' );
```

Related Information

Advanced: Query Optimization
ALTER DATABASE Statement [page 662]
sa_unload_cost_model System Procedure [page 1742]

1.6.8.56 sa_locks System Procedure

Displays all locks (including mutexes) in the database.

Syntax

```
sa_locks(
   [ connection
   [, creator
   [, table_name
   [, max_locks
   [, object_type ] ] ] ] ]
)
```

Parameters

connection

Use this optional INTEGER parameter to specify a connection ID number. The procedure returns lock information only about the specified connection. The default value is 0 (or NULL), in which case information is returned about all connections.

creator

Use this optional CHAR(128) parameter to specify a user ID. The procedure returns information only about the objects owned by the specified user. The default value for the creator parameter is NULL. When this parameter is set to NULL, sa_locks returns the following information:

- if `table_name` is not specified, locking information is returned for all objects in the database
- if `table_name` is specified, locking information is returned for objects with the specified name that were created by the current user

table_name

Use this optional CHAR(128) parameter to specify an object name. The procedure returns information only about the specified objects. The default value is NULL, in which case information is returned about all objects.

max_locks

Use this optional INTEGER parameter to specify the maximum number of locks for which to return information. The default value is 1000. The value -1 means return all lock information.

object_type

Use this optional CHAR(5) parameter to limit your results to the type of object associated with the lock. Specify *ALL* to return lock information for all object types. Specify *TABLE* to return lock information for tables, global temporary tables, and materialized views. Specify *MUTEX* to return mutex information. If you do not specify `object_type`, the procedure returns lock information for all object types.

Result Set

Column name	Data type	Description
conn_name	VARCHAR(128)	The name of the current connection.
conn_id	INTEGER	The connection ID number.
user_id	CHAR(128)	The user ID for the connection.
table_type	CHAR(6)	The type of object associated with the lock. Possible values are: • BASE, for a table • GLBTMP, for a global temporary table • MVIEW, for a materialized view • MUTEX, for a mutex
creator	VARCHAR(128)	The owner of the object.
table_name	VARCHAR(128)	The object on which the lock is held.
index_id	INTEGER	The index ID or NULL.

Column name	Data type	Description
lock_class	CHAR(16)	The lock class. Possible values are: *Schema*, *Row*, *Table*, *Position*, or NULL (for mutexes).
lock_duration	CHAR(12)	The duration of the lock. Possible values are: *Transaction*, *Position*, or *Connection*.
lock_type	CHAR(16)	The lock type.
row_identifier	UNSIGNED BIGINT	The identifier for a row. This is either an 8-byte row identifier or NULL.

Remarks

Information in the lock_type column of the result set pertains only to table locks and varies depending on the table lock classification indicated in the lock_class column. Use this table to understand the relationship between values in these two columns.

Lock class	Lock types	Comments
Schema	Shared (shared schema lock) Exclusive (exclusive schema lock)	For schema locks, the row_identifier and index ID values are NULL.

Lock class	Lock types	Comments
Row	Read (read lock) Intent (intent lock) ReadPK (read lock) Write (write lock) WriteNoPK (write lock) Surrogate (surrogate lock)	Row read locks can be short-term locks (scans at isolation level 1) or can be long-term locks at higher isolation levels. The lock_duration column indicates whether the read lock is of short duration because of cursor stability (*Position*) or long duration, held until COMMIT/ROLLBACK (*Transaction*). Row locks are always held on a specific row, whose 8-byte row identifier is reported as a 64-bit integer value in the row_identifier column. A surrogate lock is a special case of a row lock. Surrogate locks are held on surrogate entries, which are created when referential integrity checking is delayed. There is not a unique surrogate lock for every surrogate entry created in a table. Rather, a surrogate lock corresponds to the set of surrogate entries created for a given table by a given connection. The row_identifier value is unique for the table and connection associated with the surrogate lock. If required, key and non-key portions of a row can be locked independently. A connection can obtain a read lock on the key portion of a row for shared (read) access so that other connections can still obtain write locks on other non-key columns of a row. Updating non-key columns of a row does not interfere with the insertion and deletion of foreign rows referencing that row.
Table	Shared (shared table lock) Intent (intent to update table lock) Exclusive (exclusive table lock)	None

Lock class	Lock types	Comments
Position	Phantom (phantom lock) Insert (insert lock)	Usually a position lock is also held on a specific row, and that row's 64-bit row identifier appears in the row_identifier column in the result set. However, position locks can be held on entire scans (index or sequential), in which case the row_identifier column is NULL. A position lock can be associated with a sequential table scan, or an index scan. The index_id column indicates whether the position lock is associated with a sequential scan. If the position lock is held because of a sequential scan, the index_id column is NULL. If the position lock is held as the result of a specific index scan, the index identifier of that index is listed in the index_id column. The index identifier corresponds to the primary key of the ISYSIDX system table, which can be viewed using the SYSIDX view. If the position lock is held for scans over all indexes, the index ID value is -1.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR system privilege.

> **i** Note
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

You can execute the following query to identify locks.

```
CALL sa_locks( );
```

Use the sa_locks system procedure to view the locks that are currently held in the database, including information about the connection holding the lock, the lock duration, and the lock type. Execute a query that joins the results of the sa_locks system procedure to a particular table by using the ROWID of the table in the join predicate.

```
SELECT S.conn_id, S.user_id, S.lock_class, S.lock_type, E.*
  FROM sa_locks() S JOIN Employees E WITH( NOLOCK )
     ON ROWID(E) = S.row_identifier
  WHERE S.table_name = 'Employees';
```

The result set of the sa_locks system procedure contains the row_identifier column that allows you to identify the row in a table the lock refers to. It may not be necessary to specify the WITH NOLOCK clause; however, if the query is issued at isolation levels other than 0, the query may block until the locks are released, which reduces the usefulness of this method of checking.

Related Information

Table Locks
How Locking Works
Types of Locks
Schema Locks
Locks During Inserts
Row Locks
Position Locks
Viewing Locks (Interactive SQL)
SYSIDX System View [page 1916]

1.6.8.57 sa_make_object System Procedure (Deprecated)

Ensures that a skeletal instance of an object exists before executing an ALTER statement.

Syntax

```
sa_make_object(
objtype
, objname
[, owner
[, tabname ]  ]
)
```

```
objtype:
'procedure'
```

```
| 'function'
| 'view'
| 'trigger'
| 'service'
| 'event'
```

Parameters

objtype

Use this CHAR(30) parameter to specify the type of object being created. If objtype is 'trigger', this argument specifies the owner of the table on which the trigger is to be created.

objname

Use this CHAR(128) parameter to specify the name of the object to be created.

owner

Use this optional CHAR(128) parameter to specify the owner of the object to be created. The default is NULL.

tabname

This CHAR(128) parameter is required only if objtype is 'trigger', in which case you use it to specify the name of the table on which the trigger is to be created. The default is NULL.

Remarks

This procedure can be used in scripts that are run repeatedly to create or modify a database schema, however its use is deprecated in favor of using the CREATE OR REPLACE statement for the type object you are creating or modifying, wherever possible. Using CREATE OR REPLACE is more efficient, and offers the correct behavior when trying to create an object that already exists.

If you use the sa_make_object system procedure, you typically follow it by an ALTER statement that contains the entire object definition.

Privileges

You must have EXECUTE privilege on the system procedure, as well as other privileges, as follows:

Procedures or functions owned by the invoker

CREATE PROCEDURE, CREATE ANY PROCEDURE, or CREATE ANY OBJECT system privilege

Procedures or functions owned by other users

CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege

Services

MANAGE ANY WEB SERVICE system privilege

Events

MANAGE ANY EVENT or CREATE ANY OBJECT system privilege

Views owned by the invoker

CREATE VIEW, CREATE ANY VIEW, or CREATE ANY OBJECT system privilege

Views owned by other users

CREATE ANY VIEW or CREATE ANY OBJECT system privilege

Triggers

If the trigger is on a table owned by you, you must have either the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege.

If the trigger is on a table owned by another user, you must have either the CREATE ANY TRIGGER or the CREATE ANY OBJECT system privilege. Additionally, you must have one of the following:

- ALTER ANY TABLE privilege
- ALTER ANY OBJECT system privilege
- ALTER permission on the table on which the trigger is being created.

Side Effects

Automatic commit

Example

The following statements ensure that a skeleton procedure definition is created, define the procedure, and grant privileges on it. A script file containing these instructions could be run repeatedly against a database without error.

```
CALL sa_make_object( 'procedure', 'myproc' );
ALTER PROCEDURE myproc( in p1 INT, in p2 CHAR(30) )
BEGIN
    // ...
END;
GRANT EXECUTE ON myproc TO public;
```

The following example uses the sa_make_object system procedure to add a skeleton web service.

```
CALL sa_make_object( 'service', 'my_web_service' );
```

Related Information

ALTER EVENT Statement [page 675]
ALTER FUNCTION Statement [page 682]

ALTER PROCEDURE Statement [page 704]
ALTER SERVICE Statement [HTTP Web Service] [page 718]
ALTER SERVICE Statement [SOAP Web Service] [page 724]
ALTER TRIGGER Statement [page 768]
ALTER VIEW Statement [page 772]

1.6.8.58 sa_materialized_view_can_be_immediate System Procedure

Returns whether the specified materialized view can be defined as immediate.

Syntax

```
sa_materialized_view_can_be_immediate(
view_name
, owner_name
)
```

Parameters

view_name

Use this CHAR(128) parameter to specify the name of the materialized view. If `view_name` is NULL, an error is returned.

owner_name

Use this CHAR(128) parameter to specify the owner of the materialized view. If `owner_name` is NULL, the current user ID is used.

Result Set

Column name	Data type	Description
SQLStateVal	CHAR(6)	The SQLSTATE returned.
ErrorMessage	LONG VARCHAR	The error message corresponding to the SQLSTATE.

Remarks

There are restrictions on whether the specified manual view can be changed to an immediate view. Use this system procedure to determine whether the change is permitted.

Each row in the result set corresponds to a single SQLSTATE returned for a view. So, if the materialized view definition violates more than one restriction, the results include multiple rows for the view.

You can combine the output of this system procedure with the output of the sa_materialized_view_info system procedure to get information about the status of views and whether they can be made immediate.

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must be the owner of the materialized view, or have the ALTER ANY MATERIALIZED VIEW or ALTER ANY OBJECT system privilege.

Side Effects

All metadata for the specified materialized view, and all dependencies, are loaded into the server cache.

Example

Execute the following statements to create a manual view, view10, and refresh it:

```
CREATE MATERIALIZED VIEW view10
   AS (SELECT C.ID, C.Surname, sum(P.UnitPrice) as revenue, C.CompanyName,
SO.OrderDate
         FROM Customers C, SalesOrders SO, SalesOrderItems SOI, Products P
         WHERE C.ID = SO.CustomerID
         AND SO.ID = SOI.ID
         AND P.ID = SOI.ProductID
         GROUP BY C.ID, C.Surname, C.CompanyName, SO.OrderDate);
REFRESH MATERIALIZED VIEW view10;
```

Use the following query to find the reasons why view10 cannot be changed to an immediate view:

```
SELECT SQLStateVal AS "SQLstate", ErrorMessage AS Description
   FROM sa_materialized_view_can_be_immediate( 'view10', NULL )
   ORDER BY SQLSTATE;
```

Related Information

Materialized Views Restrictions
Changing the Refresh Type for a Materialized View
sa_materialized_view_info System Procedure [page 1643]
The materialized view %1 cannot be changed to immediate because it has already been initialized

The materialized view %1 cannot be changed to immediate because it does not have a unique index on non-nullable columns
The materialized view cannot be changed to immediate because COUNT(*) must be part of the SELECT list
The materialized view cannot be changed to immediate because it does not have a unique index on non-aggregate, non-nullable columns

1.6.8.59 sa_materialized_view_info System Procedure

Returns information about the specified materialized views.

Syntax

```
sa_materialized_view_info(
[ view_name
[, owner_name ] ]
)
```

Parameters

view_name

Use this optional CHAR(128) parameter to specify the name of the materialized view for which to return information. The default is NULL.

owner_name

Use this optional CHAR(128) parameter to specify the owner of the materialized view. The default is NULL.

Result Set

Column name	Data type	Description
OwnerName	CHAR(128)	The owner of the view.
ViewName	CHAR(128)	The name of the view.
Status	CHAR(1)	Status information about the view. Possible values are: **D** disabled **E** enabled

Column name	Data type	Description
DataStatus	CHAR(1)	Status information about data in the view. Possible values are: **E** An error occurred during the last refresh attempt. The view is enabled, but uninitialized. **F** The underlying tables have not changed since the last refresh, and the view is considered fresh. The view is enabled and initialized. **N** The view is uninitialized. This occurs when one of the following is true: • the view has not been refreshed since it was created • the data has been truncated from the view • the view is disabled **S** An underlying table has changed since the last refresh, and the view is considered stale. The view is enabled and initialized.
ViewLastRefreshed	TIMESTAMP	The database server local date and time when the view was last refreshed. If the value of ViewLastRefreshed is NULL, the view is uninitialized. This value is affected by simulated time zone.
DataLastModified	TIMESTAMP	For a stale view, the last database server local date and time that underlying data was modified. This value is affected by simulated time zone. The value is NULL for views that are not initialized, or for views that are not considered stale.

Column name	Data type	Description
AvailForOptimization	CHAR(1)	Information about the availability of the view for use by the optimizer. Possible values are: **D** Use by the optimizer is disabled. The owner of the view doesn't allow the view to be used by the optimizer. **I** The view cannot be used by the optimizer for some internal reason, for example its definition doesn't meet the conditions required. However, the owner has not explicitly disallowed its use by the optimizer. **N** The view contains no data because a refresh has not been done or has failed. The view can be used by the optimizer by the owner of the view, but it is not initialized. **O** There is an incompatible option value for current connection. The view can be used by the optimizer and its definition meets all the required conditions, but the current option settings are not compatible with the options settings used to create the view. **Y** The view can be used by the optimizer. The owner of the view allows the view to be used by the optimizer and the view definition meets all the conditions needed to be used by the optimizer.
RefreshType	CHAR(1)	The refresh type for the view. Possible values are: **I** The view is an immediate view. Immediate views are refreshed immediately when changes to the data in an underlying table impact the data in the materialized view. **M** The view is a manual view. Manual views are refreshed manually, for example using the REFRESH MATERIALIZED VIEW statement, or the sa_refresh_materialized_views system procedure.

Remarks

If neither `view_name` nor `owner_name` are specified or both are NULL, information about all materialized views in the database is returned.

If `owner_name` is not specified or is NULL, information about all materialized views named `view_name` is returned.

This procedure can be useful for determining the list of materialized views that will never be considered by the optimizer because of a problem with the view definition. The AvailForOptimization value is **I** for these materialized views.

The following table shows how the AvailForOptimization property is determined. Starting from the left column, you read across the row to see the conditions that must be in place to result in the value found in the AvailForOptimization column.

User allows view to be used in optimization?	The view definition satisfies all the conditions required for use?	The connection options match those required for use of the view?	The view is initialized?	AvailForOptimization value
Yes	Yes	Yes	Yes	Y
No	N/A	N/A	N/A	D
Yes	No	N/A	Yes	I
Yes	N/A	N/A	No	N
Yes	Yes	No	Yes	O

An initialized materialized view can be empty. This occurs when there is no data in the underlying tables that meets the materialized view definition. An empty view is not considered the same as an uninitialized materialized view, which also has no data in it. The value of the ViewLastRefreshed property allows you to distinguish between whether the view is uninitialized (NULL), or empty because of data in the underlying tables (non-NULL).

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

All metadata for the specified materialized views, and all dependencies, are loaded into the database server cache.

Example

The following statement returns information about all materialized views in the database:

```
SELECT *
   FROM sa_materialized_view_info();
```

The results of the sa_materialized_view_info system procedure can be combined with the results of the sa_materialized_view_can_be_immediate system procedure to return status information, and whether the view

is eligible for being an immediate view. Execute the following statements to create materialized views that are examined for this example:

```
CREATE MATERIALIZED VIEW view0 AS (
   SELECT ID, Name, Description, Size
   FROM Products
   WHERE Quantity > 0 );
CREATE UNIQUE INDEX u_view0
   ON view0( ID );
ALTER MATERIALIZED VIEW view0
   IMMEDIATE REFRESH;
CREATE MATERIALIZED VIEW view00 AS (
   SELECT ID, Name, Description, Size
   FROM Products
   WHERE Quantity <= 0 );
CREATE UNIQUE INDEX u_view00
   ON view00( ID );
CREATE MATERIALIZED VIEW view1 AS (
   SELECT ID, Name, Description, Size
   FROM Products
   WHERE Quantity = 0 );
ALTER MATERIALIZED VIEW view1
   DISABLE;
CREATE MATERIALIZED VIEW view100
   AS (SELECT C.ID, C.Surname, sum(P.UnitPrice) as revenue, C.CompanyName,
SO.OrderDate
         FROM Customers C, SalesOrders SO, SalesOrderItems SOI, Products P
         WHERE C.ID = SO.CustomerID
         AND SO.ID = SOI.ID
         AND P.ID = SOI.ProductID
         GROUP BY C.ID, C.Surname, C.CompanyName, SO.OrderDate);
REFRESH MATERIALIZED VIEW view100;
```

Execute the following statement to return the status and eligibility information for the views owned by you:

```
SELECT ViewName, Status, ViewLastRefreshed, AvailForOptimization, RefreshType,
CanBeImmediate
FROM sa_materialized_view_info() AS V,
   LATERAL( SELECT LIST( ErrorMessage, '' )
            FROM sa_materialized_view_can_be_immediate( V.ViewName,
V.OwnerName ) ) AS I( CanBeImmediate )
WHERE OwnerName  = USER_NAME();
```

ViewName	Status	ViewLastRefreshed	AvailForOptimization	RefreshType	CanBeImmediate
view0	E	(NULL)	N	I	
view00	E	(NULL)	N	M	
view1	D	(NULL)	N	M	Cannot use view 'view1' because it has been disabled

ViewName	Status	ViewLastRefreshed	AvailForOptimization	RefreshType	CanBeImmediate
view100	E	2008-02-12 16:47:00.000	Y	M	The materialized view view100 cannot be changed to immediate because it has already been initialized. The materialized view view100 cannot be changed to immediate because it does not have a unique index on non-nullable columns. The materialized view cannot be changed to immediate because COUNT(*) is required to be part of the SELECT list. The materialized view cannot be changed to immediate because it does not have a unique index on non-aggregate non-nullable columns.

The results show that:

- view0 was never refreshed and is an immediate view.

- view00 was never refreshed and is a manual view.
- view1 is disabled
- view100 is a manual view that was last refreshed at 2008-02-12 16:47:00.000.
- view00 can be changed to an immediate view because there are no error messages in the CanBeImmediate column.
- view1 and view100 cannot be changed to immediate views for the reasons listed in the CanBeImmediate column.

Related Information

Materialized Views
Materialized Views Restrictions
Materialized Views Restrictions
Whether to Set Refresh Type to Manual or Immediate
Changing the Refresh Type for a Materialized View
Determining Which Materialized Views Were Considered by the Optimizer
sa_materialized_view_can_be_immediate System Procedure [page 1641]

1.6.8.60 sa_migrate System Procedure

Migrates a set of remote tables to a SQL Anywhere database.

Syntax

```
sa_migrate(
base_table_owner
, server_name
[, table_name
[, owner_name
[, database_name
[, migrate_data
[, drop_proxy_tables
[, migrate_fkeys ] ] ] ] ] ]
)
```

Parameters

base_table_owner

Use this VARCHAR(128) parameter to specify the user on the target database who owns the migrated tables. Use the GRANT CONNECT statement to create this user. A value is required for this parameter.

server_name

Use this VARCHAR(128) parameter to specify the name of the remote server that is being used to connect to the remote database. Use the CREATE SERVER statement to create this server. A value is required for this parameter.

table_name

If you are migrating a single table, use this optional VARCHAR(128) parameter to specify the table name. Otherwise, you should specify NULL (the default) for this parameter. Do not specify NULL for both the table_name and owner_name parameters.

owner_name

If you are migrating only tables that belong to one owner, use this optional VARCHAR(128) parameter to specify the owner's name. Otherwise, you should specify NULL (the default) for this parameter. Do not specify NULL for both the table_name and owner_name parameters.

database_name

Use this optional VARCHAR(128) parameter to specify the name of the remote database. You must specify the database name to migrate tables from only one database on the remote server. Otherwise, use NULL (the default) for this parameter.

migrate_data

Use this optional BIT parameter to specify whether the data in the remote tables is migrated. This parameter can be 0 (do not migrate data) or 1 (migrate data). By default, data is migrated (1 is the default).

drop_proxy_tables

Use this optional BIT parameter to specify whether the proxy tables created for the migration process are dropped once the migration is complete. This parameter can be 0 (proxy tables are not dropped) or 1 (proxy tables are dropped). By default, the proxy tables are dropped (1 is the default).

migrate_fkeys

Use this optional BIT parameter to specify whether the foreign key mappings are migrated. This parameter can be 0 (do not migrate foreign key mappings) or 1 (migrate foreign key mappings). By default, the foreign key mappings are migrated (1 is the default).

Remarks

You can use this procedure to migrate tables to SQL Anywhere from a remote Oracle, IBM DB2, Microsoft SQL Server, Adaptive Server Enterprise, or SQL Anywhere database. This procedure allows you to migrate in one step a set of remote tables, including their foreign key mappings, from the specified server. The sa_migrate system procedure calls the following system procedures:

- sa_migrate_create_remote_table_list
- sa_migrate_create_tables
- sa_migrate_data
- sa_migrate_create_remote_fks_list
- sa_migrate_create_fks
- sa_migrate_drop_proxy_tables

You might want to use these system procedures instead of sa_migrate if you need more flexibility. For example, if you are migrating tables with foreign key relationships that are owned by different users, you cannot retain the foreign key relationships if you use sa_migrate.

Before you can migrate any tables, you must first create a remote server to connect to the remote database using the CREATE SERVER statement. You may also need to create an external login to the remote database using the CREATE EXTERNLOGIN statement.

You can migrate all the tables from the remote database to a SQL Anywhere database by specifying only the `base_table_owner` and `server_name` parameters. However, if you specify only these two parameters, all the tables that are migrated will belong to one owner in the target SQL Anywhere database. If tables have different owners on the remote database and you want them to have different owners on the SQL Anywhere database, then you must migrate the tables for each owner separately, specifying the `base_table_owner` and `owner_name` parameters each time you call the sa_migrate procedure.

> ⚠ Caution
>
> Do not specify NULL for both the `table_name` and `owner_name` parameters. Supplying NULL for both the `table_name` and `owner_name` parameters migrates all the tables in the database, including system tables. As well, tables that have the same name, but different owners in the remote database all belong to one owner in the target database. Migrate tables associated with one owner at a time.

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must have the following system privileges:

- CREATE TABLE or CREATE ANY TABLE (if you are not the base table owner)
- SELECT ANY TABLE (if you are not the base table owner)
- INSERT ANY TABLE (if you are not the base table owner)
- ALTER ANY TABLE (if you are not the base table owner)
- CREATE ANY INDEX (if you are not the base table owner)
- DROP ANY TABLE (if you are not the base table owner)

Side Effects

None

Example

The following statement migrates all the tables belonging to user DBA from the remote database, including foreign key mappings; migrates the data in the remote tables; and drops the proxy tables when migration is complete. In this example, all the tables that are migrated belong to LocalUser in the target SQL Anywhere database.

```
CALL sa_migrate( 'LocalUser', 'RemoteSA', NULL, 'DBA', NULL, 1, 1, 1 );
```

The following statement migrates all the tables that belong to user DBA from the remote database. In the target SQL Anywhere database, these tables belong to the user LocalUser. Proxy tables created during the migration are not dropped at completion and belong to LocalUser.

```
CALL sa_migrate( 'LocalUser', 'RemoteSA', NULL, 'DBA', NULL, 1, 0, 1 );
```

Related Information

Database Migration to SQL Anywhere
CREATE EXTERNLOGIN Statement [page 858]
CREATE SERVER Statement [page 962]
GRANT Statement [page 1193]
sa_migrate_create_remote_table_list System Procedure [page 1655]
sa_migrate_create_tables System Procedure [page 1657]
sa_migrate_data System Procedure [page 1659]
sa_migrate_create_remote_fks_list System Procedure [page 1654]
sa_migrate_create_fks System Procedure [page 1652]
sa_migrate_drop_proxy_tables System Procedure [page 1660]

1.6.8.61 sa_migrate_create_fks System Procedure

Creates foreign keys for each table listed in the dbo.migrate_remote_fks_list table.

Syntax

```
sa_migrate_create_fks( i_table_owner )
```

Parameters

i_table_owner

Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the migrated foreign keys. To migrate tables that belong to different user, execute this procedure for each user whose tables you want to migrate. The `i_table_owner` is created using the GRANT CONNECT statement. A value is required for this parameter.

Remarks

This procedure creates foreign keys for each table that is listed in the dbo.migrate_remote_fks_list table. The user specified by the `i_table_owner` argument owns the foreign keys in the target database.

If the tables in the target SQL Anywhere database do not all have the same owner, you must execute this procedure for each user who owns tables for which you need to migrate foreign keys.

> **i** Note
>
> This system procedure is used with several other migration system procedures, which must be executed in sequence as listed below:
>
> 1. sa_migrate_create_remote_table_list
> 2. sa_migrate_create_tables
> 3. sa_migrate_data
> 4. sa_migrate_create_remote_fks_list
> 5. sa_migrate_create_fks
> 6. sa_migrate_drop_proxy_tables
>
> As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the ALTER ANY TABLE and CREATE ANY INDEX system privileges.

Side Effects

None

Example

The first statement creates a list of foreign keys for the tables that are listed in the dbo.migrate_remote_table_list table. The second statement creates foreign keys based on the dbo.migrate_remote_fks_list table. The foreign keys belong to tables owned by the user LocalUser on the local SQL Anywhere database.

```
CALL sa_migrate_create_remote_fks_list( 'RemoteSA' );
CALL sa_migrate_create_fks( 'LocalUser' );
```

Related Information

Database Migration to SQL Anywhere
GRANT Statement [page 1193]
sa_migrate System Procedure [page 1649]
sa_migrate_create_remote_table_list System Procedure [page 1655]
sa_migrate_create_tables System Procedure [page 1657]
sa_migrate_data System Procedure [page 1659]
sa_migrate_create_remote_fks_list System Procedure [page 1654]
sa_migrate_drop_proxy_tables System Procedure [page 1660]

1.6.8.62 sa_migrate_create_remote_fks_list System Procedure

Populates the dbo.migrate_remote_fks_list table.

Syntax

```
sa_migrate_create_remote_fks_list( server_name )
```

Parameters

server_name

Use this VARCHAR(128) parameter to specify the name of the remote server that is being used to connect to the remote database. The remote server is created with the CREATE SERVER statement. A value is required for this parameter.

Remarks

This procedure populates the dbo.migrate_remote_fks_list table with a list of foreign keys that can be migrated from the remote database. You can delete rows from this table for foreign keys that you do not want to migrate.

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the sa_migrate_create_fks system procedure contains the list of migrate procedures, and the order in which you must execute them.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following statement creates a list of foreign keys for the tables that are listed in the dbo.migrate_remote_table_list table.

```
CALL sa_migrate_create_remote_fks_list( 'RemoteSA' );
```

Related Information

Database Migration to SQL Anywhere
CREATE SERVER Statement [page 962]
sa_migrate System Procedure [page 1649]
sa_migrate_create_remote_table_list System Procedure [page 1655]
sa_migrate_create_tables System Procedure [page 1657]
sa_migrate_data System Procedure [page 1659]
sa_migrate_create_fks System Procedure [page 1652]
sa_migrate_drop_proxy_tables System Procedure [page 1660]

1.6.8.63 sa_migrate_create_remote_table_list System Procedure

Populates the dbo.migrate_remote_table_list table.

Syntax

```
sa_migrate_create_remote_table_list(
i_server_name
[, i_table_name
[, i_owner_name
[, i_database_name ] ] ]
)
```

Parameters

i_server_name

Use this VARCHAR(128) parameter to specify the name of the remote server that is being used to connect to the remote database. The remote server is created with the CREATE SERVER statement. A value is required for this parameter.

i_table_name

Use this optional VARCHAR(128) parameter to specify the name(s) of the tables that you want to migrate, or NULL to migrate all the tables. The default is NULL. Do not specify NULL for both the `i_table_name` and `i_owner_name` parameters.

i_owner_name

Use this optional VARCHAR(128) parameter to specify the user who owns the tables on the remote database that you want to migrate, or NULL to migrate all the tables. The default is NULL. Do not specify NULL for both the `i_table_name` and `i_owner_name` parameters

i_database_name

Use this optional VARCHAR(128) parameter to specify the name of the remote database from which you want to migrate tables. The default is NULL. When migrating tables from Adaptive Server Enterprise and Microsoft SQL Server databases, you must specify the database name.

Remarks

This procedure populates the dbo.migrate_remote_table_list table with a list of tables that can be migrated from the remote database. You can delete rows from this table for remote tables that you do not want to migrate.

If you do not want all the migrated tables to have the same owner on the target SQL Anywhere database, you must execute this procedure for each user whose tables you want to migrate.

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

> ⚠ Caution
>
> Do not specify NULL for both the `i_table_name` and `i_owner_name` parameters. Supplying NULL for both the `i_table_name` and `i_owner_name` parameters migrates all the tables in the database, including system tables. As well, tables that have the same name, but different owners in the remote database all belong to one owner in the target database. Migrate tables associated with one owner at a time.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the sa_migrate_create_fks system procedure contains the list of migrate procedures, and the order in which you must execute them.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following statement creates a list of tables that belong to the user DBA on the remote database.

```
CALL sa_migrate_create_remote_table_list( 'RemoteSA', NULL, 'DBA', NULL );
```

Related Information

Database Migration to SQL Anywhere
CREATE SERVER Statement [page 962]
sa_migrate System Procedure [page 1649]
sa_migrate_create_tables System Procedure [page 1657]
sa_migrate_data System Procedure [page 1659]
sa_migrate_create_remote_fks_list System Procedure [page 1654]
sa_migrate_create_fks System Procedure [page 1652]
sa_migrate_drop_proxy_tables System Procedure [page 1660]

1.6.8.64 sa_migrate_create_tables System Procedure

Creates a proxy table and base table for each table listed in the dbo.migrate_remote_table_list table.

Syntax

```
sa_migrate_create_tables( i_table_owner )
```

Parameters

i_table_owner

Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the migrated tables. This user is created using the GRANT CONNECT statement. A value is required for this parameter.

Remarks

This procedure creates a base table and proxy table for each table listed in the dbo.migrate_remote_table_list table (created using the sa_migrate_create_remote_table_list procedure). These proxy tables and base tables are owned by the user specified by the `i_table_owner` argument. This procedure also creates the same primary key indexes and other indexes for the new table that the remote table has in the remote database.

If you do not want all the migrated tables to have the same owner on the target SQL Anywhere database, you must execute the sa_migrate_create_remote_table_list procedure and the sa_migrate_create_tables procedure for each user who will own migrated tables.

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the sa_migrate_create_fks system procedure contains the list of migrate procedures, and the order in which you must execute them.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the CREATE ANY TABLE system privilege.

Side Effects

None

Example

The first statement creates a list of tables that belong to the user DBA on the remote database. The second statement creates base tables and proxy tables on the target SQL Anywhere database using that list. These tables belong to the user LocalUser.

```
CALL sa_migrate_create_remote_table_list( 'RemoteSA', NULL, 'DBA', NULL );
CALL sa_migrate_create_tables( 'LocalUser' );
```

Related Information

Database Migration to SQL Anywhere
sa_migrate System Procedure [page 1649]
sa_migrate_create_remote_table_list System Procedure [page 1655]
sa_migrate_data System Procedure [page 1659]

sa_migrate_create_remote_fks_list System Procedure [page 1654]
sa_migrate_create_fks System Procedure [page 1652]
sa_migrate_drop_proxy_tables System Procedure [page 1660]

1.6.8.65 sa_migrate_data System Procedure

Migrates data from the remote database tables to the target SQL Anywhere database.

Syntax

```
sa_migrate_data( i_table_owner )
```

Parameters

i_table_owner

Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the migrated tables. This user is created using the GRANT CONNECT statement. A value is required for this parameter.

Remarks

This procedure migrates the data from the remote database to the SQL Anywhere database for tables belonging to the user specified by the `i_table_owner` argument.

When the tables on the target SQL Anywhere database do not all have the same owner, you must execute this procedure for each user whose tables have data that you want to migrate.

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the sa_migrate_create_fks system procedure contains the list of migrate procedures, and the order in which you must execute them.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SELECT ANY TABLE and INSERT ANY TABLE system privileges.

Side Effects

None

Example

The following statement migrates data to the target SQL Anywhere database for tables that belong to the user LocalUser.

```
CALL sa_migrate_data( 'LocalUser' );
```

Related Information

Database Migration to SQL Anywhere
sa_migrate System Procedure [page 1649]
sa_migrate_create_remote_table_list System Procedure [page 1655]
sa_migrate_create_tables System Procedure [page 1657]
sa_migrate_create_remote_fks_list System Procedure [page 1654]
sa_migrate_create_fks System Procedure [page 1652]
sa_migrate_drop_proxy_tables System Procedure [page 1660]

1.6.8.66 sa_migrate_drop_proxy_tables System Procedure

Drops the proxy tables that were created for migration purposes.

Syntax

```
sa_migrate_drop_proxy_tables( i_table_owner )
```

Parameters

i_table_owner

Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the proxy tables. This user is created using the GRANT CONNECT statement. A value is required for this parameter.

Remarks

This procedure drops the proxy tables that were created for the migration. The user who owns these proxy tables is specified by the `i_table_owner` argument.

If the migrated tables are not all owned by the same user on the target SQL Anywhere database, you must call this procedure for each user to drop all the proxy tables.

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the sa_migrate_create_fks system procedure contains the list of migrate procedures, and the order in which you must execute them.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the DROP ANY TABLE object-level privilege.

Side Effects

None

Example

The following statement drops the proxy tables on the target SQL Anywhere database that belong to the user LocalUser.

```
CALL sa_migrate_drop_proxy_tables( 'LocalUser' );
```

Related Information

Database Migration to SQL Anywhere
sa_migrate System Procedure [page 1649]
sa_migrate_create_remote_table_list System Procedure [page 1655]
sa_migrate_create_tables System Procedure [page 1657]
sa_migrate_data System Procedure [page 1659]
sa_migrate_create_remote_fks_list System Procedure [page 1654]
sa_migrate_create_fks System Procedure [page 1652]

1.6.8.67 sa_mirror_server_status System Procedure

Returns the connection status of the current servers and all the servers that are directly or indirectly receiving log pages from the current server. On primary servers, the procedure returns the status of all connected servers.

Syntax

```
sa_mirror_server_status( )
```

Result Set

Column name	Data type	Description
server_name	CHAR(128)	The name of the server.
state	CHAR(20)	The connection status of the server. It can be one of the following values: • connected • disconnected In some circumstances, a disconnected server has a status of connected. When this situation occurs, the last_updated time is in the past.
last_updated	TIMESTAMP WITH TIME ZONE	The time the server status was last updated.
load_current	DOUBLE	The amount of work that the database server is currently performing.
load_last_1_min	DOUBLE	The amount of work that the database server has performed in the previous minute.
load_last_5_mins	DOUBLE	The amount of work that the database server has performed in the previous 5 minutes.
load_last_10_mins	DOUBLE	The amount of work that the database server has performed in the previous 10 minutes.
num_connections	UNSIGNED INTEGER	The number of connections to the database server.
num_processors	UNSIGNED INTEGER	The number of database server processors.
log_written	UNSIGNED BIGINT	The latest transaction log position written to disk based on the last update received from the server.

Column name	Data type	Description
log_applied	UNSIGNED BIGINT	The last operation from the transaction log that has been applied based on the last update received from the server. This value is the same as the value of the CurrentRedoPos property

Remarks

Each server updates its status and that of its copy nodes every 5 seconds. On mirror servers, the procedure returns the status of any copy nodes that are receiving log pages from the mirror server, but does not return the status of the primary server. The columns with the prefix **load** represent a computed load on the database server. The value returned represents the database server load, and not the load from other processes. Higher load values indicate that the database server has more work to perform.

The time in the last_updated column is that of the server in the server_name column. The state is accurate for the time returned in the last_updated column.

When the NodeType connection parameter is specified, the database server uses load information to redirect connections. The database server selects the mirror server with the lowest load; if all servers have the same load, the server with the fewest connections is used.

If a copy node was shut down at approximately the same time as its parent, the procedure may still report that the copy node is connected for several minutes. However, the last_updated column remains the same, indicating that the copy node has not reported updated status messages and is likely disconnected.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns the connection status of the current servers and all the servers that are directly or indirectly receiving log pages from the current server when mirroring is in effect.

```
CALL sa_mirror_server_status( );
```

Related Information

Database Mirroring
NodeType (NODE) Connection Parameter
CREATE MIRROR SERVER Statement [page 901]
ALTER MIRROR SERVER Statement [page 695]
DROP MIRROR SERVER Statement [page 1105]

1.6.8.68 sa_nchar_terms System Procedure

Breaks an NCHAR string into terms and returns each term as a row along with its position.

Syntax

```
sa_nchar_terms(
text
[, config_name
[, owner ] ]
)
```

Parameters

text

The LONG NVARCHAR string you are parsing.

config_name

Use this optional CHAR(128) parameter to specify the text configuration object to apply when processing the string. The default is 'default_nchar'.

owner

Use this optional CHAR(128) parameter to specify the owner of the text configuration object. The default value is NULL. The current user is assumed if the owner is not specified or if NULL is specified.

Remarks

You can use this system procedure to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following statement returns the terms in the NCHAR string, "It's a work-at-home day!", using the default NCHAR text configuration object, default_nchar:

```
CALL sa_nchar_terms (N'It''s a work-at-home day!', 'default_nchar', 'sys');
```

term	position
It	1
s	2
a	3
work	4
at	5
home	6
day	7

Related Information

Full Text Search
Text Configuration Object Concepts and Reference
sa_char_terms System Procedure [page 1534]

1.6.8.69 sa_performance_diagnostics System Procedure

Returns a summary of request timing information for all connections when the database server has request timing logging enabled.

Syntax

```
sa_performance_diagnostics( )
```

Result Set

Column name	Data type	Description
Number	INTEGER	Returns the connection ID (a number) for the current connection.
Name	VARCHAR(255)	Returns the name of the current connection. You can specify a connection name using the ConnectionName (CON) connection parameter. The following names are used for temporary connections created by the database server: • INT:ApplyRecovery • INT:BackupDB • INT:Checkpoint • INT:Cleaner • INT:CloseDB • INT:CreateDB • INT:CreateMirror • INT:DelayedCommit • INT:DiagRcvr • INT:DropDB • INT:EncryptDB • INT:Exchange • INT:FlushMirrorLog • INT:FlushStats • INT:HTTPReq • INT:PromoteMirror • INT:PurgeSnapshot • INT:ReconnectMirror • INT:RecoverMirror • INT:RedoCheckpoint • INT:RefreshIndex • INT:ReloadTrigger • INT:RenameMirror • INT:RestoreDB • INT:StartDB • INT:VSS
Userid	VARCHAR(255)	Returns the user ID for the connection.
DBNumber	INTEGER	Returns the ID number of the database.
LoginTime	TIMESTAMP	Returns the date and time the connection was established. This value is affected by simulated time zone.

Column name	Data type	Description
TransactionStartTime	TIMESTAMP	Returns the date and time the database was first modified after a COMMIT or ROLLBACK, or an empty string if no modifications have been made to the database since the last COMMIT or ROLLBACK. This value is affected by simulated time zone.
LastReqTime	TIMESTAMP	Returns the date and time at which the last request for the specified connection started. This column can contain null for internal connections, such as events. This value is affected by simulated time zone.
ReqType	VARCHAR(255)	Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CONNECT_POOL_CACHE.
ReqStatus	VARCHAR(255)	Returns the status of the request. It can be one of the following values: **Idle** The connection is not currently processing a request. **Unscheduled*** The connection has work to do and is waiting for an available database server worker. **BlockedIO*** The connection is blocked waiting for an I/O. **BlockedContention*** The connection is blocked waiting for access to shared database server data structures. **BlockedLock** The connection is blocked waiting for a locked object. **Executing** The connection is executing a request. The values marked with an asterisk (*) are only returned when logging of request timing information has been turned on for the database server using the -zt server option. If request timing information is not being logged (the default), the values are reported as Executing.
ReqTimeUnscheduled	DOUBLE	Returns the amount of unscheduled time, or NULL if the -zt option was not specified.
ReqTimeActive	DOUBLE	Returns the amount of time, in seconds, spent processing requests, or NULL if the -zt option was not specified.
ReqTimeBlockIO	DOUBLE	Returns the amount of time, in seconds, spent waiting for I/O to complete, or NULL if the -zt option was not specified.
ReqTimeBlockLock	DOUBLE	Returns the amount of time, in seconds, spent waiting for a lock, or NULL if the -zt option was not specified.

Column name	Data type	Description
ReqTimeBlockContention	DOUBLE	Returns the amount of time, in seconds, spent waiting for atomic access, or NULL if the RequestTiming server property is set to Off.
ReqCountUnscheduled	INTEGER	Returns the number of times the connection waited for scheduling, or NULL if the -zt option was not specified.
ReqCountActive	INTEGER	Returns the number of requests processed, or NULL if the RequestTiming server property is set to Off.
ReqCountBlockIO	INTEGER	Returns the number of times the connection waited for I/O to complete, or NULL if the -zt option was not specified.
ReqCountBlockLock	INTEGER	Returns the number of times the connection waited for a lock, or NULL if the -zt option was not specified.
ReqCountBlockContention	INTEGER	Returns the number of times the connection waited for atomic access, or NULL if the -zt option was not specified.
LastIdle	INTEGER	Returns the number of ticks between requests.
BlockedOn	INTEGER	Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.
UncommitOp	INTEGER	Returns the number of uncommitted operations.
CurrentProcedure	VARCHAR(255)	Returns the name of the procedure that a connection is currently executing. If the connection is executing nested procedure calls, the name is the name of the current procedure. If there is no procedure executing, an empty string is returned.
EventName	VARCHAR(255)	Returns the name of the associated event if the connection is running an event handler. Otherwise, an empty string is returned.
CurrentLineNumber	INTEGER	Returns the current line number of the procedure or compound statement a connection is executing. The procedure can be identified using the CurrentProcedure property. If the line is part of a compound statement from the client, an empty string is returned.

Column name	Data type	Description
LastStatement	LONG VARCHAR	Returns the most recently prepared SQL statement for the current connection. The LastStatement value is set when a statement is prepared, and is cleared when a statement is dropped. Only one statement string is remembered for each connection. If sa_conn_activity reports a non-empty value for a connection, it is most likely the statement that the connection is currently executing. If the statement had completed, it would likely have been dropped and the property value would have been cleared. If an application prepares multiple statements and retains their statement handles, then the LastStatement value does not reflect what a connection is currently doing. When client statement caching is enabled, and a cached statement is reused, this property returns an empty string.
LastPlanText	LONG VARCHAR	Returns the long text plan of the last query executed on the connection. You control the remembering of the last plan by setting the RememberLastPlan option of the sa_server_option system procedure, or using the -zp server option.
AppInfo	LONG VARCHAR	Returns information about the client that made the connection. For HTTP connections, this includes information about the browser. For connections using older versions of jConnect or SAP Open Client, the information may be incomplete. The API value can be DBLIB, ODBC, OLEDB, ADO.NET, iAnywhereJDBC, CAPI_JavaScript-XS, CAPI_Node.js, CAPI_PerlDBD, CAPI_PHP, CAPI_PYTHON, CAPI_RUBY, DBEXPRESS, or any other user-defined value.
LockCount	INTEGER	Returns the number of locks held by the connection.
SnapshotCount	INTEGER	Returns the number of snapshots associated with the connection.

Remarks

The sa_performance_diagnostics system procedure returns a result set consisting of a set of request timing properties and statistics if the server has been told to collect the information. Recording of request timing information must be turned on the database server before calling sa_performance_diagnostics. To do this, specify the -zt option when starting the database server or execute the following:

```
CALL sa_server_option( 'RequestTiming','ON' );
```

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR system privilege.

> **i** Note
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

Find all requests that are currently executing, and have been executing for more than 60 seconds:

```
SELECT  Number, Name,
      CAST( DATEDIFF( second, LastReqTime, CURRENT TIMESTAMP ) AS DOUBLE ) AS
ReqTime
FROM  sa_performance_diagnostics()
WHERE ReqStatus <> 'IDLE' AND ReqTime > 60.0
ORDER BY ReqTime DESC;
```

Related Information

Advanced Topic: Temporary Connections
-zt Database Server Option
-zt Database Server Option
sa_performance_statistics System Procedure [page 1671]
sa_server_option System Procedure [page 1698]
ConnectionName (CON) Connection Parameter

1.6.8.70 sa_performance_statistics System Procedure

Returns performance statistics for the server, databases, and connections.

Syntax

```
sa_performance_statistics( )
```

Result Set

Column name	Data type	Description
DBNumber	INTEGER	Returns the ID number of the database. Returns NULL if the property type is Server.
ConnNumber	INTEGER	Returns the connection ID (a number) for the current connection. Returns NULL if the Type is Server or Database.
PropNum	INTEGER	Returns the property number.
PropName	VARCHAR(255)	Returns the property name.
Value	BIGINT	Returns the property value.

Remarks

The sa_performance_statistics system procedure returns a result set consisting of a set of performance statistics. The results are a subset of the results you can return using the PROPERTY, DB_PROPERTY, and CONNECTION_PROPERTY functions.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR system privilege.

Note

Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

The following example unloads all performance statistics to a text file named `dump_stats.txt`:

```
UNLOAD
   SELECT CURRENT TIMESTAMP, *
   FROM sa_performance_statistics()
   TO 'dump_stats.txt'
   APPEND ON;
```

Related Information

sa_performance_diagnostics System Procedure [page 1665]
sa_server_option System Procedure [page 1698]

1.6.8.71 sa_post_login_procedure System Procedure

Determines whether a user's password is about to expire.

Syntax

sa_post_login_procedure( )

Parameters

None

Result Set

Column name	Data type	Description
message_text	VARCHAR(255)	If message_action is 1, message_text returns the message to display. If message_action is 0, message_text is NULL.
message_action	INTEGER	Whether the password is about to expire (1=yes, 0=no).

Remarks

The sa_post_login_procedure system procedure is the default setting for the post_login_procedure database option.

sa_post_login_procedure uses the user's password_life_time and password_grace_time login policy option values, and the current date and time, to determine whether a user's password is about to expire. If it is, the message to display to the user is returned in the result set.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example uses sa_post_login_procedure to determine whether the current user's password is about to expire:

```
CALL sa_post_login_procedure();
```

Related Information

Login Policies

1.6.8.72 sa_procedure_profile System Procedure

Reports information about the execution time for each line within procedures, functions, events, or triggers that have been executed in a database.

Syntax

```
sa_procedure_profile(
   [ filename
   [, save_to_file ] ]
)
```

Parameters

filename

Use this optional LONG VARCHAR parameter to specify the file to which the profiling information should be saved, or from which file it should be loaded. The default is NULL. See the Remarks section below for more about saving and loading the profiling information.

save_to_file

Use this optional INTEGER parameter to specify whether to save the profiling information to a file, or load it from a previously stored file. The default is 0.

Result Set

Column name	Data type	Description
object_type	CHAR(1)	The type of object. See the Remarks section below for a list of possible object types.
object_name	CHAR(128)	The name of the stored procedure, function, event, or trigger. If the object_type is C or D, then this is the name of the foreign key for which the system trigger was defined.
owner_name	CHAR(128)	The object's owner.
table_name	CHAR(128)	The table associated with a trigger (the value is NULL for other object types).
line_num	UNSIGNED INTEGER	The line number within the procedure.

Column name	Data type	Description
executions	UNSIGNED INTEGER	The number of times the line has been executed.
millisecs	UNSIGNED INTEGER	The time to execute the line, in milliseconds.
percentage	DOUBLE	The percentage of the total execution time required for the specific line.
foreign_owner	CHAR(128)	The database user who owns the foreign table for a system trigger.
foreign_table	CHAR(128)	The name of the foreign table for a system trigger.

Remarks

You can use this procedure to:

Return detailed procedure profiling information

To do this, you can simply call the procedure without specifying any arguments.

Save detailed procedure profiling information to file

To do this, you must include the `filename` argument and specify 1 for the `save_to_file` argument.

Load detailed procedure profiling information from a previously saved file

To do this, you must include the `filename` argument and specify 0 for the `save_to_file` argument. When using the procedure in this way, the loaded file must have been created by the same database as the one from which you are running the procedure; otherwise, the results may be unusable.

Since the result set includes information about the execution times for individual lines within procedures, triggers, functions, and events, and what percentage of the total procedure execution time those lines use, you can use this profiling information to fine-tune slower procedures that may decrease performance.

Before you can profile your database, you must enable profiling.

The object_type column of the result set can be:

P

stored procedure

F

function

E

event

T

trigger

C

ON UPDATE system trigger

D

ON DELETE system trigger

If you want summary information instead of line by line details for each execution, use the sa_procedure_profile_summary procedure instead.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR or MANAGE PROFILING system privilege.

You must also have the following privileges.

- SELECT ANY TABLE (when `filename` is not NULL and `save_to_file` is 1)
- LOAD ANY TABLE (when `filename` is not NULL and `save_to_file` is 0)

> **i Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

The following statement returns the execution time for each line of every procedure, function, event, or trigger that has been executed in the database:

```
CALL sa_procedure_profile( );
```

The following statement returns the same detailed procedure profiling information as the example above, and saves it to a file called `detailedinfo.txt`:

```
CALL sa_procedure_profile( 'detailedinfo.txt', 1 );
```

Either of the following statements can be used to load detailed procedure profiling information from a file called `detailedinfo.txt`:

```
CALL sa_procedure_profile( 'detailedinfo.txt', 0 );
```

```
CALL sa_procedure_profile( 'detailedinfo.txt' );
```

Related Information

Running a Comprehensive Profiling Session (Profiler)
sa_server_option System Procedure [page 1698]
sa_procedure_profile_summary System Procedure [page 1677]

1.6.8.73 sa_procedure_profile_summary System Procedure

Reports summary information about the execution times for all procedures, functions, events, or triggers that have been executed in a database.

Syntax

```
sa_procedure_profile_summary(
   [ filename
   [, save_to_file ] ]
)
```

Parameters

filename

Use this optional LONG VARCHAR parameter to specify the file to which the profiling information is saved, or from which file it should be loaded. The default is NULL. See the Remarks section below for more about saving and loading the profiling information.

save_to_file

Use this optional INTEGER parameter to specify whether to save the summary information to a file, or to load it from a previously saved file. The default is 0.

Result Set

Column name	Data type	Description
object_type	CHAR(1)	The type of object. See the Remarks section below for a list of possible object types.
object_name	CHAR(128)	The name of the stored procedure, function, event, or trigger.
owner_name	CHAR(128)	The object's owner.

Column name	Data type	Description
table_name	CHAR(128)	The table associated with a trigger (the value is NULL for other object types).
executions	UNSIGNED INTEGER	The number of times each procedure has been executed.
millisecs	UNSIGNED INTEGER	The time to execute the procedure, in milliseconds.
foreign_owner	CHAR(128)	The database user who owns the foreign table for a system trigger.
foreign_table	CHAR(128)	The name of the foreign table for a system trigger.

Remarks

You can use this procedure to:

Return current summary information

To do this, you can simply call the procedure without specifying any arguments.

Save current summary information to file

To do this, you must include the `filename` argument and specify 1 for the `save_to_file` argument.

Load stored summary information from a file

To do this, you must include the `filename` argument and specify 0 for the `save_to_file` argument. When using the procedure in this way, the loaded file must have been created by the same database as the one from which you are running the procedure; otherwise, the results may be unusable.

Since the procedure returns information about the usage frequency and efficiency of stored procedures, functions, events, and triggers, you can use this information to fine-tune slower procedures to improve database performance.

Before you can profile your database, you must enable profiling.

The object_type column of the result set can be:

P

stored procedure

F

function

E

event

T

trigger

C

ON UPDATE system trigger

D

ON DELETE system trigger

If you want line by line details for each execution instead of summary information, use the sa_procedure_profile procedure instead.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR or MANAGE PROFILING system privilege.

You must also have the following privileges.

- SELECT ANY TABLE (when `filename` is not NULL and `save_to_file` is 1)
- LOAD ANY TABLE (when `filename` is not NULL and `save_to_file` is 0)

> **i Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

The following statement returns the execution time for any procedure, function, event, or trigger that has been executed in the database:

```
CALL sa_procedure_profile_summary( );
```

The following statement returns the same summary information as the previous example, and saves it to a file called `summaryinfo.txt`:

```
CALL sa_procedure_profile_summary( 'summaryinfo.txt', 1 );
```

Either of the following statements can be used to load stored summary information from a file called `summaryinfo.txt`:

```
CALL sa_procedure_profile_summary( 'summaryinfo.txt', 0 );
```

```
CALL sa_procedure_profile_summary( 'summaryinfo.txt' );
```

Related Information

Running a Comprehensive Profiling Session (Profiler)
sa_server_option System Procedure [page 1698]
sa_procedure_profile System Procedure [page 1674]

1.6.8.74 sa_recompile_views System Procedure

Locates view definitions stored in the catalog that do not have column definitions and causes the column definitions to be created.

Syntax

```
sa_recompile_views( [ ignore_errors ] )
```

Parameters

ignore_errors

Use this optional INTEGER parameter to specify whether to return errors during the recompilation. If you specify 0, an error is returned for each view for which column definition failed. If you specify 1, or any value other than 0, no errors are returned. The default is 0.

Remarks

This procedure is used to locate views in the catalog that do not have column definitions and execute an ALTER VIEW statement with the RECOMPILE clause to create the column definitions. The procedure does this for each view that does not have a column definition until there are none left that require compilation or until any remaining column definitions cannot be created. If the procedure is unable to recompile any views, an error is reported. Errors can be suppressed by specifying a non-zero parameter to this procedure.

Caution

The sa_recompile_views system procedure should only be called from within a `reload.sql` script. This procedure is used by the Unload utility (dbunload) and should not be used explicitly.

The sa_recompile_views system procedure does not attempt to recompile materialized views or any view marked DISABLED.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the ALTER ANY VIEW system privilege.

Side Effects

For each regular view that does not have a VALID status, an `ALTER VIEW owner.viewname ENABLE` statement is executed, causing an automatic commit.

Example

The following example from a `reload.sql` script uses the sa_recompile_views system procedure to locate view definitions stored in the catalog that do not have column definitions and cause the column definitions to be created. Errors are ignored.

```
CALL sa_recompile_views( 1 );
```

Related Information

Statuses for Regular Views
force_view_creation Option (Reserved for System Use)
ALTER VIEW Statement [page 772]

1.6.8.75 sa_refresh_materialized_views System Procedure

Initializes all materialized views that are in an uninitialized state.

Syntax

```
sa_refresh_materialized_views( [ ignore_errors ] )
```

Parameters

ignore_errors

Use this optional INTEGER parameter to specify whether to return errors during the recompilation. If you specify 0, an error is returned for each view for which column definition failed. If you specify 1, or any value other than 0, no errors are returned. The default is 0.

Remarks

A materialized view may be in an uninitialized state because it has just been created, has just been re-enabled, or the last attempt to initialize or refresh it failed due to an error. The sa_refresh_materialized_views system procedure scans the database for all such materialized views and attempts to initialize them. If the procedure encounters an error initializing a materialized view, it continues on attempting to process the remaining uninitialized views.

You can also use the REFRESH MATERIALIZED VIEW statement to initialize a materialized view.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the ALTER ANY MATERIALIZED VIEW system privilege.

Side Effects

None

Example

The following example uses the sa_refresh_materialized_views system procedure to initialize all materialized views that are in an uninitialized state. Errors are ignored.

```
CALL sa_refresh_materialized_views( 1 );
```

Related Information

Refreshing a Materialized View Manually
REFRESH MATERIALIZED VIEW Statement [page 1321]

1.6.8.76 sa_refresh_text_indexes System Procedure

Refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH.

Syntax

```
sa_refresh_text_indexes( )
```

Remarks

The sa_refresh_text_indexes system procedure refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH. It does not refresh text indexes defined as IMMEDIATE REFRESH (the default) because changes to those indexes are made when data is changed in the underlying table.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the CREATE ANY INDEX or ALTER ANY OBJECT system privilege.

Side Effects

Automatic commit

Example

The following statement refreshes all MANUAL and AUTO REFRESH text indexes in the database:

```
CALL sa_refresh_text_indexes();
```

Related Information

Full Text Search
Text Configuration Object Concepts and Reference
DROP TEXT INDEX Statement [page 1136]
REFRESH TEXT INDEX Statement [page 1325]
TRUNCATE Statement [page 1442]

SYSTEXTIDX System View [page 1960]
sa_text_index_stats System Procedure [page 1735]
sa_text_index_vocab System Procedure [page 1737]

1.6.8.77 sa_remove_tracing_data System Procedure (Deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The sa_remove_tracing_data system procedure permanently deletes from the diagnostic tracing tables all records pertaining to the specified logging (tracing) session ID.

Syntax

```
sa_remove_tracing_data( log_session_id )
```

Parameters

log_session_id

Use this UNSIGNED INTEGER parameter to specify the ID of the logging session for which to remove the data.

Remarks

If there are no records for the specified `log_session_id`, the procedure has no effect. The procedure has no return values.

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must have the DIAGNOSTICS system role, and the MANAGE PROFILING system privilege.

Side Effects

Causes a commit upon completion, even if no records were found for the specified `log_session_id`.

Example

This example permanently deletes from the diagnostic tracing tables all records pertaining to the logging session ID 1.

```
CALL sa_remove_tracing_data( 1 );
```

Related Information

SQL Anywhere Profiler
Diagnostic Tracing Tables (Deprecated) [page 1494]

1.6.8.78 sa_report_deadlocks System Procedure

Retrieves information about deadlocks from an internal buffer created by the database server.

Syntax

```
sa_report_deadlocks( )
```

Result Set

Column name	Data type	Description
snapshotId	BIGINT	The deadlock instance (all rows pertaining to a particular deadlock have the same ID).
snapshotAt	TIMESTAMP	The database server local date and time when the deadlock occurred. This value is affected by simulated time zone.
waiter	INTEGER	The connection handle of the waiting connection.
who	VARCHAR(128)	The user ID associated with the connection that is waiting.

Column name	Data type	Description
what	LONG VARCHAR	The command being executed by the waiting connection. This information is only available if you have turned on capturing of the most recently prepared SQL statement by specifying the -zl option on the database server command line or have turned this feature on using the sa_server_option system procedure.
object_id	UNSIGNED BIGINT	The object ID of the table containing the row. If the deadlock involves a mutex, the value is the object ID of the mutex involved in the deadlock.
record_id	BIGINT	The row ID of the associated row.
owner	INTEGER	The connection handle of the connection owning the lock being waited on.
is_victim	BIT	Identifies the rolled back transaction.
rollback_operation_count	UNSIGNED INTEGER	The number of uncommitted operations that may be lost if the transaction rolls back.
table_id	UNSIGNED BIGINT	The table ID associated with the object being locked.

Remarks

When the log_deadlocks option is set to On, the database server logs information about deadlocks in an internal buffer. You can view the information in the log using the sa_report_deadlocks system procedure.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR system privilege.

> **i Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

You can execute the following query to identify deadlocks.

```
CALL sa_report_deadlocks( );
```

Related Information

System Events
Example: Determining Who Is Blocked in a Deadlock (SQL)
log_deadlocks Option
sa_server_option System Procedure [page 1698]
-zl Database Server Option
sa_server_option System Procedure [page 1698]

1.6.8.79 sa_reserved_words System Procedure

Returns a list of reserved words. Many of the keywords that appear in SQL statements are reserved words.

Syntax

```
sa_reserved_words( )
```

Result Set

Column name	Data type	Description
reserved_word	CHAR(128)	A reserved word.

Remarks

The procedure takes no parameters and returns one word per row. The list of reserved words is based on the version of the database server that executes the query, not the version of the software used to create the database file.

Not all words in the list may be enabled as reserved words. For example, the keyword LIMIT may be enabled or disabled as a reserved word using the reserved_keywords option.

Privileges

None

Side Effects

You must have EXECUTE privilege on the system procedure.

Example

The following statement returns a list of reserved words:

```
SELECT * FROM sa_reserved_words();
```

Related Information

Reserved Words [page 6]

1.6.8.80 sa_reset_identity System Procedure

Allows the next identity value to be set for a table. Use this procedure to change the AUTOINCREMENT value for the next row that will be inserted.

Syntax

```
sa_reset_identity(
tbl_name
[, owner_name
[, new_identity ] ]
```

```
)
```

Parameters

tbl_name

Use this CHAR(128) parameter to specify the table for which you want to reset the identity value. If the owner of the table is not specified, `tbl_name` must uniquely identify a table in the database.

owner_name

Use this optional CHAR(128) parameter to specify the owner of the table for which you want to reset the identity value. The default is NULL. If `owner_name` is not specified, then use a named parameter value for the third argument. For example:

```
CALL sa_reset_identity( 'Employees', new_identity=100 );
```

new_identity

Use this optional BIGINT parameter to specify the non-negative, non-NULL value from which you want the autoincrementing to start. The default is 0.

Remarks

The next identity value generated for a row inserted into the table is `new_identity` + 1.

No checking occurs to see whether `new_identity` + 1 conflicts with existing rows in the table. For example, if you specify `new_identity` as 100, the next row inserted gets an identity value of 101. However, if 101 already exists in the table, the row insertion fails.

The sa_reset_identity system procedure cannot be used on a table having no columns with a default of either AUTOINCREMENT or GLOBAL AUTOINCREMENT.

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must be the owner of the table, or have the ALTER ANY TABLE system privilege.

Side Effects

Causes a checkpoint to occur after the value has been updated

Example

The following statement resets the next identity value to 101:

```
CALL sa_reset_identity( 'Employees', 'GROUPO', 100 );
```

Related Information

The AUTOINCREMENT Default
The GLOBAL AUTOINCREMENT Default

1.6.8.81 sa_rowgenerator System Procedure

Returns a result set with rows between a specified start and end value.

Syntax

```
sa_rowgenerator(
  [ rstart
  [, rend
  [, rstep ] ] ]
)
```

Parameters

rstart

Use this optional INTEGER parameter to specify the starting value. The default value is 0.

rend

Use this optional INTEGER parameter to specify the ending value that is greater than or equal to `rstart`. The default value is 100.

rstep

Use this optional INTEGER parameter to specify the increment by which the sequence values are increased. The default value is 1.

Result Set

Column name	Data type	Description
row_num	INTEGER	Sequence number.

Remarks

The sa_rowgenerator procedure can be used in the FROM clause of a query to generate a sequence of numbers. This procedure is an alternative to using the RowGenerator system table. You can use sa_rowgenerator for such tasks as:

- generating test data for a known number of rows in a result set.
- generating a result set with rows for values in every range. For example, you can generate a row for every day of the month, or you can generate ranges of ZIP codes.
- generating a query that has a specified number of rows in the result set. This may be useful for testing the performance of queries.

No rows are returned if you do not specify correct start and end values and a positive non-zero step value.

You can emulate the behavior of the RowGenerator table with the following statement:

```
SELECT row_num FROM sa_rowgenerator( 1, 255 );
```

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following query returns a result set containing one row for each day of the current month.

```
SELECT DATEADD( day, row_num-1,
        YMD( DATEPART( year, CURRENT DATE ),
            DATEPART( month, CURRENT DATE ), 1 ) )
    AS day_of_month
    FROM sa_rowgenerator( 1, 31, 1 )
    WHERE DATEPART( month, day_of_month ) = DATEPART( month, CURRENT DATE )
```

```
    ORDER BY row_num;
```

The following query shows how many employees live in ZIP code ranges (0-9999), (10000-19999), ..., (90000-99999). Some of these ranges have no employees, which causes a warning.

The sa_rowgenerator procedure can be used to generate these ranges, even though no employees have a ZIP code in the range.

```
SELECT row_num AS r1, row_num+9999 AS r2, COUNT( PostalCode ) AS zips_in_range
    FROM sa_rowgenerator( 0, 99999, 10000 ) D LEFT JOIN Employees
        ON PostalCode BETWEEN r1 AND r2
    GROUP BY r1, r2
    ORDER BY 1;
```

The following example generates 10 rows of data and inserts them into the NewEmployees table:

```
INSERT INTO NewEmployees ( ID, Salary, Name )
    SELECT row_num, CAST( RAND() * 1000 AS INTEGER ), 'Mary'
    FROM sa_rowgenerator( 1, 10 );
```

The following example uses the sa_rowgenerator system procedure to create a view containing all integers. The value 2147483647 in this example represents the maximum signed integer that is supported.

```
CREATE VIEW Integers AS
    SELECT row_num AS n
    FROM sa_rowgenerator( 0, 2147483647, 1 );
```

This example uses the sa_rowgenerator system procedure to create a view containing dates from 0001-01-01 to 9999-12-31. The value 3652058 in this example represents the number of days between 0001-01-01 and 9999-12-31, the earliest and latest dates that are supported.

```
CREATE VIEW Dates AS
    SELECT DATEADD( day, row_num, '0001-01-01' ) AS d
    FROM sa_rowgenerator( 0, 3652058, 1 );
```

The following query returns all years between 1900 and 2058 that have 54 weeks.

```
SELECT DATEADD ( day, row_num, '1900-01-01' ) AS d, DATEPART ( week, d ) w
    FROM sa_rowgenerator ( 0, 63919, 1 )
    WHERE w = 54;
```

Related Information

RowGenerator Table (dbo) [page 1511]

Null value eliminated in aggregate function

1.6.8.82 sa_save_trace_data System Procedure

Saves tracing data to base tables.

Syntax

```
sa_save_trace_data( )
```

Remarks

While a tracing session is running, diagnostic data is stored in temporary versions of the diagnostic tracing tables. When you stop a tracing session, you specify whether you want to permanently store the tracing data in the base tables for diagnostic tracing. If you do not choose to save the data, you can still save the data after the session is stopped by using the sa_save_trace_data system procedure.

The sa_save_trace_data system procedure returns an error if tracing is still in progress; you must stop tracing to use this system procedure.

The sa_save_trace_data system procedure can be used even if the user specified WITHOUT SAVING when stopping tracing. Also, the procedure must be called from the tracing database.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE PROFILING system privilege.

Side Effects

Automatic commit.

Example

This example saves tracing data to the diagnostic tracing tables.

```
CALL sa_save_trace_data( );
```

Related Information

Diagnostic Tracing Tables (Deprecated) [page 1494]

1.6.8.83 sa_send_udp System Function

Sends a UDP packet to the specified address.

Syntax

```
sa_send_udp(
destAddress
, destPort
, msg
)
```

Parameters

destAddress

Use this CHAR(254) to specify either the host name or IP number.

destPort

Use this UNSIGNED SMALLINT parameter to specify the port number to use.

msg

Use this LONG BINARY parameter to specify the message to send to the specified address. If this value is a string, it must be enclosed in single quotes.

Returns

This function returns an INTEGER status code.

Remarks

This function sends a single UDP packet to the specified address. The function returns 0 if the message is sent successfully, and returns an error code if an error occurs. The error code is one of the following:

- -1 if the message is too large to send over a UDP socket (as determined by the operating system) or if there is a problem with the destination address
- the Winsock/Posix error code that is returned by the operating system

If the `msg` parameter contains binary data or is more complex than a string, you may want to use a BINARY variable. For example:

```
CREATE VARIABLE v LONG BINARY;
SET v='This is a UDP message';
SELECT sa_send_udp( '10.25.99.124', 1234, v );
DROP VARIABLE v;
```

This function can be used with MobiLink server-initiated synchronization to wake up the MobiLink Listener utility (`dblsn.exe`). If you use the sa_send_udp function as a way to notify the MobiLink Listener, you should append a 1 to your UDP packet. This number is a server-initiated synchronization protocol number. In future versions of MobiLink, new protocol versions may cause the MobiLink Listener to behave differently.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY WEB SERVICE system privilege.

Side Effects

None

Example

The following example sends the message "This is a test" to IP address 10.25.99.196 on port 2345:

```
SELECT sa_send_udp( '10.25.99.196', 2345, 'This is a test' );
```

Related Information

Advanced: Sending a Push Notification Using the sa_send_udp System Procedure

1.6.8.84 sa_server_messages System Procedure

Allows you to return messages from the database server messages window as a result set.

Syntax

```
sa_server_messages(
```

```
[ first_msg
[, num_msgs ] ]
)
```

Parameters

first_msg

Use this optional UNSIGNED BIGINT parameter to specify the ID of the first or last message to be returned, depending on the sign of the `num_msgs` parameter. The default is NULL, which means that the search starts at the beginning of the list if `num_msgs` is NULL or non-negative; the search starts past the end of the list if `num_msgs` is negative.

num_msgs

Use this optional BIGINT parameter to specify the number of messages to be returned. The sign indicates whether the request is for messages starting at `first_msg` or ending at `first_msg`. The default is NULL, which means that all messages starting at `first_msg` to the end of the list are returned.

Result Set

Column name	Data type	Description
msg_id	UNSIGNED BIGINT	Unique message ID. Message IDs start at 0.
msg_text	LONG VARCHAR	Message text.
msg_time	TIMESTAMP	Database server local date and time when the message was issued. This value is affected by simulated time zone.
msg_severity	VARCHAR(255)	Message severity. This column contains one of the following values: **INFO** Informational message. **WARN** Warning. **ERR** Error.

Column name	Data type	Description
msg_category	VARCHAR(255)	Message category. This column contains one of the following values: **STARTUP/SHUTDOWN** Messages related to database server or database startup or shutdown. **CHKPT** Messages related to checkpoints. **MSG** Messages generated using the MESSAGE or PRINT statements. **DBA_MSG** Messages generated using the MESSAGE statement that would have required the SERVER OPERATOR system privilege, such as messages sent to the event log. **CONN** Messages about database server connectivity. **OTHER** All other types of messages.
msg_database	VARCHAR(255)	Database name associated with the message if it applies to one specific database. Otherwise, NULL.

Remarks

When new messages are sent to the console, old messages with the same category or severity are deleted if the number of messages exceeds the value of the MessageCategoryLimit property. As a result, there may be gaps in the result set, and two consecutive rows may not have consecutive message IDs.

The STARTUP/SHUTDOWN message category does not show shutdown messages for servers. Shutdown messages are shown only if multiple databases are running on a server and one or more are shut down.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following command requests 100 messages starting at the message whose ID is 3:

```
CALL sa_server_messages( 3, 100 );
```

The following command requests 500 messages up to, and including, message 4032:

```
CALL sa_server_messages( 4032, -500 );
```

The following commands request all messages starting with message 3:

```
CALL sa_server_messages( 3, NULL );
```

```
CALL sa_server_messages( 3 );
```

The following command requests the first 100 messages in the list:

```
CALL sa_server_messages( NULL, 100 );
```

The following command requests the last 100 messages in the list:

```
CALL sa_server_messages( NULL, -100 );
```

The following commands request all the messages in the list:

```
CALL sa_server_messages( NULL, NULL );
```

```
CALL sa_server_messages( );
```

1.6.8.85 sa_server_option System Procedure

Overrides a database server option while the database server is running.

Syntax

```
sa_server_option(
opt
, val
)
```

Parameters

opt

Use this CHAR(128) parameter to specify a database server option name.

val

Use this LONG VARCHAR parameter to specify the new value for the database server option.

Remarks

Use this procedure to override some database server options temporarily, without restarting the database server.

The option values that are changed using this procedure are reset to their default values when the database server shuts down. To change an option value every time you start the database server, specify the corresponding database server option when the database server is started (if one exists).

The following option settings can be changed. Default values are shown in bold:

Option name	Values	Additional information
AutoMultiProgrammingLevel	**YES**, NO	When set to YES, the database server automatically adjusts its multiprogramming level, which controls the maximum number of tasks that can be active at a time. If you choose to control the multiprogramming level manually by setting this option to NO, you can still set the initial, minimum, and maximum values for the multiprogramming level.
AutoMultiProgrammingLevelStatistics	YES, **NO**	When set to YES, statistics for automatic multiprogramming level adjustments appear in the database server message log.
CacheSizingStatistics	YES, **NO**	When set to YES, display cache information in the database server messages window whenever the cache size changes.

Option name	Values	Additional information
CockpitDB	To start or stop the Cockpit: 'DBF=filename | DBF=AUTO**[; START=ON** | START=OFF**]**' To create a copy of the Cockpit database: **'SAVETO=**filename[; **ACTION=CONTINUE** | ACTION=SWITCH]'	Use the CockpitDB option to start or stop the Cockpit, or change the Cockpit database. Specify at least one parameter. The DBF and START parameters cannot be specified with the SAVETO and ACTION parameters. • Use the DBF parameter to specify the Cockpit database for the Cockpit to use. The Cockpit configuration settings are saved to this file. If you specify a file that does not exist, then it is created. Specify the full path for the file. Otherwise if you use a relative path, it is read relative to the current working directory (or if disk sandboxing is enabled, then the relative path is read relative to the directory where the main database file is located). The value specified with the DBF parameter becomes the default DBF value until it is changed or the database server is shut down. Set DBF=AUTO to run the Cockpit using a temporary database. When the Cockpit stops, its temporary database is deleted, so any changes made within the Cockpit are not saved. The DBF parameter can only be specified when the Cockpit is not running. When the DBF parameter is specified, the Cockpit starts unless START=OFF is also specified. When START is set to ON, the Cockpit starts. When START is set to OFF, the Cockpit stops. • Use the SAVETO parameter to create a Cockpit database that contains the settings of the currently running Cockpit. Specify the full path for the file. Otherwise if you use a relative path, it is read relative to the current working directory (or if disk sandboxing is enabled, then the relative path is read relative to the directory where the main database file is located). Set ACTION=SWITCH to have the Cockpit use this new Cockpit immediately start using this database. Otherwise, when ACTION=CONTINUE is specified, the Cockpit continues using the current Cockpit database. To start the Cockpit, execute `CALL sa_server_option('CockpitDB' 'START=ON;DBF=filename');`. To stop the Cockpit, execute `CALL sa_server_option('CockpitDB' 'START=OFF');`. To change the default file name for the Cockpit, first stop the Cockpit, and then execute `CALL sa_server_option('CockpitDB 'START=OFF;DBF=filename);` to define the new default file. To create a backup copy of the Cockpit database, execute `CALL sa_server_option ('CockpitDB' 'SAVETO=c:\file.db');`. To create a new Cockpit database and have the Cockpit use this file, execute `CALL sa_server_option ('CockpitDB' 'SAVETO=c:\file.db;ACTION=SWITCH');`.
CollectStatistics	**YES**, NO	When set to YES, the database server collects Performance Monitor statistics. When set through the sa_server_option to NO, will disable statement performance data collection.

Option name	Values	Additional information
ConnsDisabled	YES, **NO**	When set to YES, no other connections are allowed to any databases on the database server.
ConnsDisabled-ForDB	YES, **NO**	When set to YES, no other connections are allowed to the current database.
ConsoleLogFile	`filename`	The name of the file used to record database server message log information. Specifying an empty string stops logging to the file. Double any backslash characters in the path because this value is a SQL string.
ConsoleLogMax-Size	`file-size`, in bytes	The maximum size, in bytes, of the file used to record database server message log information. When the database server message log file reaches the size specified by either this property or the -on server option, the file is renamed with the extension `.old` appended (replacing an existing file with the same name if one exists). The database server message log file is then restarted.
CurrentMulti-ProgrammingLe-vel	Integer. Default is **20**.	Sets the multiprogramming level of the database server.
Databa-seCleaner	**ON**, OFF	Do not change the setting of this option except on the recommendation of Technical Support.
DeadlockLog-ging	ON, **OFF**, RE-SET, CLEAR	Controls deadlock logging. The value deadlock_logging is also supported. The following values are supported: **ON** Enables deadlock logging. **OFF** Disables deadlock logging and leaves the deadlock data available for viewing. **RESET** Clears the logged deadlock data, if any exists, and then enables deadlock logging. **CLEAR** Clears the logged deadlock data, if any exists, and then disables deadlock logging. Once deadlock logging is enabled, you can use the sa_report_deadlocks system procedure to retrieve deadlock information from the database.
DebuggingInfor-mation	YES, **NO**	Displays diagnostic messages and other messages for troubleshooting purposes. The messages appear in the database server messages window.
DiskSandbox	ON, **OFF**	Sets the default disk sandbox settings for all databases started on the database server that do not have explicit disk sandbox settings. Changing the disk sandbox settings by using the sa_server_option system procedure does not affect databases already running on the database server. To use the sa_server_option system procedure to change disk sandbox settings, you must provide the secured feature key for the manage_disk_sandbox feature.

Option name	Values	Additional information
DropBadStatistics	**YES**, NO	Allows automatic statistics management to drop statistics that return bad estimates from the database.
DropUnusedStatistics	**YES**, NO	Allows automatic statistics management to drop statistics that have not been used for 90 consecutive days from the database.
IdleTimeout	Integer, in minutes. The default is **240**.	Disconnects TCP/IP connections that have not submitted a request for the specified number of minutes. This prevents inactive connections from holding locks indefinitely. The minimum value is 0 (no timeout) and the maximum value is 32767.
IPAddressMonitorPeriod	Integer, in seconds. The default is **120** for portable devices, **0** otherwise.	Sets the time to check for new IP addresses in seconds. The minimum value is 10 and the default is 0. For portable devices, the default value is 120 seconds.
LivenessTimeout	Integer, in seconds. The default is **120**.	A liveness packet is sent periodically across a client/server TCP/IP network to confirm that a connection is intact. If the network server runs for a LivenessTimeout period without detecting a liveness packet, the communication is severed. The minimum value is 0 (no timeout) and the maximum value is 32767.
MaxMultiProgrammingLevel	Integer. The default is four times the value for CurrentMultiProgrammingLevel	Sets the maximum database server multiprogramming level.
MessageCategoryLimit	Integer. The default is **400**.	Sets the minimum number of messages of each severity and category that can be retrieved using the sa_server_messages system procedure.
MiniDumpType	**NORMAL**, FULL	Controls the crash dump type. The following values are supported: **NORMAL** Enables a mini crash dump. **FULL (Windows only)** Enable a full crash dump.
MinMultiProgrammingLevel	Integer. The default is the minimum of the value of the -gtc server option and the number of logical CPUs on the computer.	Sets the minimum database server multiprogramming level.

Option name	Values	Additional information
OptionWatchAction	**MESSAGE**, ERROR	Specifies the action that the database server takes when an attempt is made to set an option in the list. The supported values are MESSAGE and ERROR. When OptionWatchAction is set to MESSAGE, and an option specified by OptionWatchList is set, a message appears in the database server messages window indicating that the option being set is on the options watch list. When OptionWatchAction is set to ERROR, an error is returned indicating that the option cannot be set because it is on the options watch list. You can view the current setting for this property by executing the following query: `SELECT DB_PROPERTY( 'OptionWatchAction' );`

OptionWatchList	Comma-separated list of database options.	Specifies a comma-separated list of database options that you want to be notified about, or have the database server return an error for, when they are set. The string length is limited to 128 bytes. By default, it is an empty string. For example, the following command adds the automatic_timestamp, float_as_double, and tsql_hex_constant option to the list of options being watched:
		
		`CALL sa_server_option( 'OptionWatchList','automatic_timestamp, float_as_double,tsql_hex_constant' );`
		
		You can view the current setting for this property by executing the following query:
		
		`SELECT DB_PROPERTY( 'OptionWatchList' );`

ProcedureProfiling	ON, **OFF**, RESET, CLEAR	ON enables procedure profiling. OFF disables procedure profiling. RESET clears profiling history and enables procedure profiling. CLEAR clears profiling history and disables procedure profiling. See the ProcedureProfiling database property description for more information.
ProcessorAffinity	Comma-delimited list of processor numbers and/or ranges. The default is that all processors are used or the setting of the -gta option.	Instructs the database server which logical processors to use on Windows or Linux. Specify a comma-delimited list of processor numbers and/or ranges. If the lower endpoint of a range is omitted, then it is assumed to be zero. If the upper endpoint of a range is omitted, then it is assumed to be the highest CPU available to the operating system. The in_use column returned by the sa_cpu_topology system procedure contains the current processor affinity of the database server, the in_use column indicates whether the database server is using a processor, and the user_selected column indicates which physical processors were specified by the -gta database server option or the ProcessorAffinity server property. Alternatively, you can query the value of the ProcessorAffinity database server property. The database server might not use all of the specified logical processors in the following cases: • If one or more of the specified logical processors does not exist, or is offline. • If the license does not allow it. If you specify an invalid processor ID, then sa_server_option returns an error.

Option name	Values	Additional information
ProfileFilterConn	`connection-id`	Instructs the database server to capture profiling information for a specific connection ID, without preventing other connections from using the database. When connection filtering is enabled, the value returned for `SELECT PROPERTY( 'ProfileFilterConn' )` is the connection ID of the connection being monitored. If no ID has been specified, or if connection filtering is disabled, the value returned is -1.
ProfileFilterUser	`user-id`	Instructs the database server to capture profiling information for a specific user ID.
PropertyHistoryList	**ON**, OFF, NONE, comma-delimited list of database server properties	**ON** When PropertyHistoryList is turned on, a default list of properties is tracked. The default is ON. **OFF** When PropertyHistoryList is turned off, property tracking is disabled for the database server. **NONE** Setting this property to NONE enables property tracking but no properties are tracked by the database server. Only properties requested by databases are tracked. **Comma-delimited list of database server properties** Specify a comma-delimited list of database server properties to track.
PropertyHistorySize	`time`, `memory-size`, MAX, DEFAULT	Specifies either the minimum amount of time to store tracked property values or the maximum amount of memory to use to store tracked property values. To set this property to a time, use the format '[HH:]MM:SS'. To set this property to a memory size, specify the memory size in bytes. For example, 1M. The default value is '00:10:00' (ten minutes), unless that amount of time violates the maximum size limit, in which case MAX is used as the default. Specify MAX to request that the fixed maximum amount of memory is used. The maximum memory is either 2% of the cache or 256 MB, whichever is smaller. When PropertyHistoryList is set, the amount of memory used for tracking property history is updated. If it has increased beyond the fixed maximum amount of memory allowed for storing property history, then an error is raised. If the amount of memory used has decreased and there is no longer enough memory to track the specified properties' history, then an error is raised. If an error is raised, then the property history reverts back to its previous value. If there is insufficient memory to track all properties specified by PropertyHistoryList, an error is returned and the property is not added to the list of tracked properties.
QuittingTime	Valid date and time	Instructs the database server to shut down at the specified time.
RememberLastPlan	YES, **NO**	Instructs the database server to capture the long text plan of the last query executed on the connection. This setting is also controlled by the -zp server option. When RememberLastPlan is turned on, obtain the textual representation of the plan of the last query executed on the connection by querying the value of the LastPlanText connection property: `SELECT CONNECTION_PROPERTY( 'LastPlanText' );`

Option name	Values	Additional information

RememberLast-Statement	YES, **NO**	Instructs the database server to capture the most recently prepared SQL statement for each database running on the server. For stored procedure calls, only the outermost procedure call appears, not the statements within the procedure.
		
		When RememberLastStatement is turned on, you can obtain the current value of the LastStatement for a connection by querying the value of the LastStatement connection property:
		
		`SELECT CONNECTION_PROPERTY( 'LastStatement' );`
		
		When client statement caching is enabled, and a cached statement is reused, this property returns an empty string.
		
		When RememberLastStatement is turned on, the following statement returns the most recently prepared statement for the specified connection:
		
		`SELECT CONNECTION_PROPERTY( 'LastStatement', connection-id );`
		
		The sa_conn_activity system procedure returns this same information for all connections.
		
		⚠ Caution
		
		When -zl is specified, or when the RememberLastStatement server setting is turned on, any user can call the sa_conn_activity system procedure or obtain the value of the LastStatement connection property to find out the most recently prepared SQL statement for any other user. Use this option with caution and turn it off when it is not required.

RequestFilter-Conn	`connection-id`, -1	Filter the request logging information so that only information for a particular connection is logged. This filtering can reduce the size of the request log file when monitoring a database server with many active connections or multiple databases. You can obtain the connection ID by executing the following:
		
		`CALL sa_conn_info( );`
		
		To log a specific connection once you have obtained the connection ID, execute the following statement:
		
		`CALL sa_server_option( 'RequestFilterConn', connection-id );`
		
		Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:
		
		`CALL sa_server_option( 'RequestFilterConn', -1 );`

Option name	Values	Additional information

RequestFilterDB	database-id, -1	Filter the request logging information so that only information for a particular database is logged. This can help reduce the size of the request log file when monitoring a server with multiple databases. You can obtain the database ID by executing the following statement when you are connected to the desired database:
		`SELECT CONNECTION_PROPERTY( 'DBNumber' );`
		To log only information for a particular database, execute the following statement:
		`CALL sa_server_option( 'RequestFilterDB', database-id );`
		Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:
		`CALL sa_server_option( 'RequestFilterDB', -1 );`

RequestLogFile	filename	The name of the file used to record request information. Specifying an empty string stops logging to the request log file. If request logging is enabled, but the request log file was not specified or has been set to an empty string, the server logs requests to the database server messages window. Double any backslash characters in the path because this value is a SQL string. When client statement caching is enabled, set the max_client_statements_cached option to 0 to disable client statement caching while the request log is captured, if the log will be analyzed using the `tracetime.pl` Perl script.

Option name	Values	Additional information
RequestLogging	SQL, HOSTVARS, PLAN, PROCEDURES, TRIGGERS, OTHER, BLOCKS, REPLACE, ALL, YES, **NONE**, NO	This call turns on logging of individual SQL statements sent to the database server for use in troubleshooting with the database server -zr and -zo options. Values can be combinations of the following, separated by either a plus sign (+), or a comma: **PLAN** enables logging of execution plans (short form). If logging of procedures (PROCEDURES) is enabled, execution plans for procedures are also recorded. **HOSTVARS** enables logging of host variable values. If you specify HOSTVARS, the information listed for SQL is also logged. **PROCEDURES** enables logging of statements executed from within procedures. **TRIGGERS** enables logging of statements executed from within triggers. **OTHER** enables logging of additional request types not included by SQL, such as FETCH and PREFETCH. However, if you specify OTHER but do not specify SQL, it is the equivalent of specifying SQL+OTHER. Including OTHER can cause the request log file to grow rapidly and could negatively impact server performance. **BLOCKS** enables logging of details showing when a connection is blocked and unblocked on another connection. **REPLACE** at the start of logging, the existing request log is replaced with a new (empty) one of the same name. Otherwise, the existing request log is opened and new entries are appended to the end of the file. **ALL** logs all supported information. This value is equivalent to specifying SQL+PLAN+HOSTVARS+PROCEDURES+TRIGGERS+OTHER+BLOCKS. This setting can cause the request log file to grow rapidly and could negatively impact server performance. **NO or NONE** turns off logging to the request log. You can view the current setting for this property by executing the following query: `SELECT PROPERTY( 'RequestLogging' );`

Option name	Values	Additional information
RequestLogMaxSize	`file-size`, in bytes	The maximum size of the file used to record request logging information, in bytes. If you specify 0, then there is no maximum size for the request logging file, and the file is never renamed. This value is the default. When the request log file reaches the size specified by either the sa_server_option system procedure or the -zs server option, the file is renamed with the extension `.old` appended (replacing an existing file with the same name if one exists). The request log file is then restarted.
RequestLogNumFiles	Integer	The number of request log file copies to retain. If request logging is enabled over a long period, the request log file can become large. The -zn option allows you to specify the number of request log file copies to retain.
RequestTiming	YES, **NO**	Instructs the database server to maintain timing information for each new connection. This feature is turned off by default. When it is turned on, the database server maintains cumulative timers for all new connections that indicate how much time the connection spent in the server in each of several states. The change is only effective for new connections, and lasts for the duration each connection. You can use the sa_performance_diagnostics system procedure to obtain a summary of this timing information, or you can retrieve individual values by inspecting the following connection properties: • ReqCountUnscheduled • ReqTimeUnscheduled • ReqCountActive • ReqTimeActive • ReqCountBlockIO • ReqTimeBlockIO • ReqCountBlockLock • ReqTimeBlockLock • ReqCountBlockContention • ReqTimeBlockContention When the RequestTiming server property is on, there is a small overhead for each request to maintain the additional counters.

Option name	Values	Additional information

SecureFeatures	`feature-list`	Allows you to manage secured features for a database server that is already running. The `feature-list` is a comma-separated list of feature names or feature sets. By adding a feature to the list, you limit its availability. To remove items from the list of secured features, specify a minus sign (-) before the feature name.
		
		For example, to secure two features, use the following syntax:
		
		`CALL sa_server_option('SecureFeatures', 'console_log,webclient_log' );`
		
		After executing this statement, the list of secured features is set according to what has been changed.
		
		To secure the LOCAL feature set, but exclude the LOCAL_IO feature subset, use the following syntax:
		
		`CALL sa_server_option('SecureFeatures', 'local,-local_io' );`
		
		To call sa_server_option('SecureFeatures', ...), the connection must have the MANAGE_FEATURES feature enabled on the connection. The -sf database server option secures MANAGE_FEATURES by default. If the current user/connection that set the SecureFeatures option does not have a feature explicitly enabled, and the list of features was disabled by that connection, then the connection is immediately affected.
		
		Any changes you make to allow or prevent access to features take effect immediately for the database server. The connection that executes the sa_server_option system procedure may or may not be affected, depending on the secured feature key the connection is using and whether it allows the connection access to the specified features.

SingleCLRInstanceVersion	35, 40, 45	This option specifies whether or not the database server uses one CLR external environment for all databases running on the database server. The option value indicates which version to start. **i Note** This option cannot be changed once the database server starts a CLR external environment.
SingleJVMLocation	`filename`	This option specifies the location of the Java VM that the database server is using for all databases running on the database server. This option can only be set if the database server has been started with the -sjvm database server option, or the UseSingleJVMInstance database server option is set to ON.
StatisticsCleaner	**ON**, OFF	The statistics cleaner fixes statistics that give bad estimates by performing scans on tables. By default the statistics cleaner runs in the background and has a minimal impact on performance. Turning off the statistics cleaner does not disable the statistic governor, but when the statistics cleaner is turned off, statistics are only created or fixed when a query is run.

Option name	Values	Additional information
TopologyAware-Scheduling	**ON**, OFF	Turns topology-aware scheduling on or off. Topology-aware scheduling causes tasks to use a single core per socket before attempting to use another core on the same socket (tasks are scheduled to use one thread per core before attempting to use other threads on the same core).
UseSingleJV-MInstance	ON, **OFF**	This option specifies whether or not the database server uses one Java VM for all databases running on the database server. **i Note** This option cannot be changed once the database server starts a Java VM.
WebClientLog-File	`filename`	The name of the web service client log file. The web service client log file is truncated each time you use the -zoc server option or the WebClientLogFile property to set or reset the file name. Double any backslash characters in the path because this value is a string.
WebClientLog-ging	ON, **OFF**	This option enables and disables logging of web service clients. The information that is logged includes HTTP requests and response data. Specify ON to start logging to the web service client log file, and specify OFF to stop logging to the file.

Privileges

You must have EXECUTE privilege on the system procedure.

You must have the MANAGE PROFILING system privilege to use the following options, which are related to request logging:

- ProcedureProfiling
- ProfileFilterConn
- ProfileFilterUser
- RequestFilterConn
- RequestFilterDB
- RequestLogFile
- RequestLogging
- RequestLogMaxSize
- RequestLogNumFiles

For all other options, your must have the SERVER OPERATOR system privilege.

You must also have the MANAGE ANY EXTERNAL ENVIRONMENT system privilege to use the SingleJVMLocation option.

Side Effects

None

Example

The following statement causes cache information to be displayed in the database server messages window whenever the cache size changes:

```
CALL sa_server_option( 'CacheSizingStatistics', 'YES' );
```

The following statement disallows new connections to the current database:

```
CALL sa_server_option( 'ConnsDisabledForDB', 'YES' );
```

The following statement enables logging of all SQL statements, procedure calls, plans, blocking and unblocking events, and starts a new request log:

```
CALL sa_server_option( 'RequestLogging', 'SQL+PROCEDURES+BLOCKS+PLAN+REPLACE' );
```

Related Information

Database Server Startup Options
Database Server Configuration of the Multiprogramming Level
Procedure Profiling (System Procedures)
Creating Secured Feature Keys
Monitoring Option Settings
List of Database Server Properties
List of Connection Properties
Database Startup Options
sa_performance_diagnostics System Procedure [page 1665]
log_deadlocks Option
sa_db_option System Procedure [page 1566]

1.6.8.86 sa_set_http_header System Procedure

Permits a web service to set an HTTP response header.

Syntax

```
sa_set_http_header(
fldname
, val
```

```
[, instance ]
)
```

Parameters

fldname

Use this CHAR(128) parameter to specify a string containing the name of one of the HTTP header fields.

val

Use this LONG VARCHAR parameter to specify the value to which the named parameter should be set. Setting a response header to NULL, effectively removes it.

instance

Use this UNSIGNED INT parameter to specify which instance of the HTTP response header to set. The default is 1.

Remarks

Setting the special header field @HttpStatus sets the status code returned with the request. The status code is also known as the response code. For example, the following script sets the status code to 404 Not Found:

```
CALL sa_set_http_header( '@HttpStatus', '404' );
```

You can create a user-defined status message by specifying a three digit status code with an optional colon-delimited text message. For example, the following script outputs a status code with the message "999 User Code":

```
CALL sa_set_http_header( '@HttpStatus', '999:User Code' );
```

> **i** Note
>
> A user defined status text message is not translated into a database character-set when logged using the LogOptions protocol option.

The body of the error message is inserted automatically. Only valid HTTP error codes can be used. Setting the status to an invalid code causes a SQL error.

The sa_set_http_header procedure always overwrites the existing header value of the header field when called.

Response headers generated automatically by the database server can be removed. For example, the following command removes the Expires response header:

```
CALL sa_set_http_header( 'Expires', NULL );
```

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example sets the Set-Cookie header field to type=chocolate and specifies the third instance of the header.

```
CALL sa_set_http_header( 'Set-Cookie', 'type=chocolate', 3 );
```

Related Information

Web Services Functions [page 221]
NEXT_HTTP_RESPONSE_HEADER Function [Web Service] [page 474]
HTTP_RESPONSE_HEADER Function [Web Service] [page 414]
Web Services System Procedures [page 1512]
LogOptions (LOPT) Protocol Option

1.6.8.87 sa_set_http_option System Procedure

Permits a web service to set an HTTP option for process control.

Syntax

```
sa_set_http_option(
optname
, val
)
```

Parameters

optname

Use this CHAR(128) parameter to specify a string containing the name of one of the HTTP options.

The supported options are:

CharsetConversion

Use this option to control whether the result set is to be automatically converted from the character set encoding of the database to the character set encoding of the client. The only permitted values are ON and OFF. The default value is ON.

AcceptCharset

Use this option to specify the web server's preferences for a response character set encoding. One or more character set encodings may be specified in order of preference. The syntax for this option conforms to the syntax used for the HTTP Accept-Charset request-header field specification in RFC2616 Hypertext Transfer Protocol.

An HTTP client such as a web browser may provide an Accept-Charset request header which specifies a list of character set encodings ordered by preference. Optionally, each encoding may be given an associated quality value (q=`qvalue`) which represents the client's preference for that encoding. By default, the quality value is 1 (q=1). Here is an example:

```
Accept-Charset: iso-8859-5, utf-8;q=0.8
```

A plus sign (+) in the AcceptCharset HTTP option value may be used as a shortcut to represent the current database character set encoding. The plus sign also indicates that the database character set encoding should take precedence if the client also specifies the encoding in its list, regardless of the quality value assigned by the client.

An asterisk (*) in the AcceptCharset HTTP option may be used to indicate that the web service should use a character set encoding preferred by the client, as long as it is also supported by the server, when client and server do not have an intersecting list.

When sending the response, the first character set encoding preferred by both client and web service is used. The client's order of preference takes precedence. If no mutual encoding preference exists, then the web service's most preferred encoding is used, unless an asterisk (*) appears in the web service list in which case the client's most preferred encoding is used.

If the AcceptCharset HTTP option is not used, the most preferred character set encoding specified by the client and supported by the server is used. If none of the encodings specified by the client are supported (or the client does not send an Accept-Charset request header) then the database character set encoding is used.

If a client does not send an Accept-Charset header then one of the following actions are taken:

- If the AcceptCharset HTTP option has not been specified then the web server will use the database character set encoding.
- If the AcceptCharset HTTP option has been specified then the web server will use its most preferred character set encoding.

If a client does send an Accept-Charset header then one of the following actions are taken:

- If the AcceptCharset HTTP option has not been specified then the web server will attempt to use one of the client's preferred character set encodings, starting with the most preferred encoding. If the web server does not support any of the client's preferred encodings, it will use the database character set encoding.

- If the AcceptCharset HTTP option has been specified then the web server will attempt to use the first preferred character set encoding common to both lists, starting with the client's most preferred encoding. For example, if the client sends an Accept-Charset header listing, in order of preference, encodings iso-a, iso-b, and iso-c and the web server prefers iso-b, then iso-a, and finally iso-c, then iso-a will be selected.

  ```
  Web client: iso-a, iso-b, iso-c
  Web server: iso-b, iso-a, iso-c
  ```

  If the intersection of the two lists is empty, then the web server's first preferred character set is used. From the following example, encoding iso-d will be used.

  ```
  Web client: iso-a, iso-b, iso-c
  Web server: iso-d, iso-e, iso-f
  ```

  If an asterisk ('*') was included in the AcceptCharset HTTP option, then emphasis would be placed on the client's choice of encodings, resulting in iso-a being used. Essentially, the use of an asterisk guarantees that the intersection of the two lists will not be empty.

The ideal situation occurs when both client and web service use the database character set encoding since this eliminates the need for character set translation and improves the response time of the web server.

If the CharsetConversion option has been set to OFF, then AcceptCharset processing is not performed.

SessionID

Use this option to create, delete or rename an HTTP session. The database connection is persisted when a web service sets this option to create an HTTP session but sessions are not persisted across server restarts. If already within a session context, this call will rename the session to the new session ID. When called with a NULL value, the session will be deleted when the web service terminates.

The generated session keys are limited to 128 characters in length and unique across databases if multiple databases are loaded.

SessionTimeout

Use this option to specify the amount of time, in minutes, that the HTTP session persists during inactivity. This timeout period is reset whenever an HTTP request uses the given session. The session is automatically deleted when the SessionTimeout is exceeded. When an HTTP session is created, the default timeout period is taken from the current setting of the http_session_timeout database option. The SessionTimeout option can be used to override this default. Allowed values are 1 to 525600 (365 days).

val

Use this LONG VARCHAR parameter to specify the value to which the named option should be set.

Remarks

Use this procedure within statements or procedures that handle web services to set options.

When sa_set_http_option is called from within a procedure invoked through a web service, and either the option or option value is invalid, an error is returned.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example illustrates the use of sa_set_http_option to indicate the web service's preference for database character set encoding. The UTF-8 encoding is specified as a second choice. The asterisk (*) indicates that the web service is willing to use the character set encoding most preferred by the client, provided that it is supported by the web server.

```
CALL sa_set_http_option( 'AcceptCharset', '+,UTF-8,*');
```

The following example illustrates the use of sa_set_http_option to correctly identify the character encoding in use by the web service. In this example, the web server is connected to a 1251CYR database and is prepared to serve HTML documents containing the Cyrillic alphabet to any web browser.

```
CREATE OR REPLACE PROCEDURE cyrillic_html()
RESULT (html_doc XML)
BEGIN
  DECLARE pos INT;
  DECLARE charset VARCHAR(30);
  CALL sa_set_http_option( 'AcceptCharset', 'iso-8859-5, utf-8' );
  SET charset = CONNECTION_PROPERTY( 'CharSet' );
  -- Change any IANA labels like ISO_8859-5:1988
  -- to ISO_8859-5 for Firefox.
  SET pos = LOCATE( charset, ':' );
  IF pos > 0 THEN
    SET charset = LEFT( charset, pos - 1 );
  END IF;
  CALL sa_set_http_header( 'Content-Type', 'text/html; charset=' ||
        charset );
  SELECT  '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' ||
    XMLCONCAT(
      XMLELEMENT('HTML',
        XMLELEMENT('HEAD',
          XMLELEMENT('TITLE', 'Cyrillic characters')
        ),
        XMLELEMENT('BODY',
          XMLELEMENT('H1', 'First 5 lowercase Russian letters'),
          XMLELEMENT('P', UNISTR('\u0430\u0431\u0432\u0433\u0434'))
        )
      )
    );
END;
CREATE SERVICE cyrillic
TYPE 'RAW'
AUTHORIZATION OFF
USER DBA
AS CALL cyrillic_html();
```

To illustrate the process of establishing the correct character set encoding to use, consider the following Accept-Charset header delivered by a web browser such as Firefox to the web service. It indicates that the browser prefers ISO-8859-1 and UTF-8 encodings but is willing to accept others.

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

The web service will not accept the ISO-8859-1 character set encoding since the web page to be transmitted contains Cyrillic characters. The web service prefers ISO-8859-5 or UTF-8 encodings as indicated by the call to sa_set_http_option. In this example, the UTF-8 encoding will be chosen since it is agreeable to both parties. The database connection property CharSet indicates which encoding has been selected by the web service. The sa_set_http_header procedure is used to indicate the HTML document's encoding to the web browser.

```
Content-Type: text/html; charset=UTF-8
```

If the web browser does not specify an Accept-Charset, then the web service defaults to its first preference, ISO-8859-5. The sa_set_http_header procedure is used to indicate the HTML document's encoding.

```
Content-Type: text/html; charset=ISO_8859-5
```

The following example sets a unique HTTP session identifier:

```
BEGIN
  DECLARE sessionid VARCHAR(30);
  DECLARE tm TIMESTAMP;
  SET tm = NOW(*);
  SET sessionid = 'MySessions_' ||
      CONVERT( VARCHAR, SECONDS(tm)*1000 + DATEPART(millisecond,tm));
  SELECT sessionid;
  CALL sa_set_http_option('SessionID', sessionid);
END;
```

The following example sets the timeout for an HTTP session to 5 minutes:

```
CALL sa_set_http_option('SessionTimeout', 5);
```

Related Information

Character Set Conversion Considerations
HTTP Session Management on an HTTP Server
Web Services Functions [page 221]
List of Connection Properties
Web Services System Procedures [page 1512]

1.6.8.88 sa_set_soap_header System Procedure

Permits the setting of SOAP headers for SOAP responses. This procedure is used within stored procedures called from SOAP web services.

Syntax

```
sa_set_soap_header(
fldname
, val
)
```

Parameters

fldname

Use this CHAR(128) parameter to specify the header key, a unique string used to reference the given header entry (it need not be identical to the localname of the `val`).

val

Use this LONG VARCHAR parameter to specify the raw XML of a top level header entry and its children within the scope of a SOAP Header element.

Remarks

All SOAP header entries set with this procedure are serialized within the SOAP Header element when the SOAP response message is sent. A `val` of NULL is not serialized. If no header entries exist for a SOAP response, then an enclosing Header element, within the SOAP envelope, is not created.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example sets the SOAP header welcome to Hello:

```
CALL sa_set_soap_header( 'welcome', '<welcome>Hello</welcome>' )
```

Related Information

Web Services Functions [page 221]
Tutorial: Using a Database Server to Access a SOAP/DISH Service
Web Services System Procedures [page 1512]

1.6.8.89 sa_set_tracing_level System Procedure (Deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. Initializes the level of tracing information to be stored in the diagnostic tracing tables.

Syntax

```
sa_set_tracing_level(
level
[, specified_scope
[, specified_name
[, do_commit ] ] ]
)
```

Parameters

level

Use this INTEGER parameter to specify the level of diagnostic tracing to perform. Possible values include:

0

Do not generate any tracing data. This level keeps the tracing session open, but does not send any tracing data to the diagnostic tracing tables.

1

Sets a basic level of tracing.

2

Sets a medium level of tracing.

3

Sets a high level of tracing.

specified_scope

Use this optional LONG VARCHAR parameter to specify the tracing scope; for example, USER, DATABASE, CONNECTION_NAME, TRIGGER, and so on. The default is NULL.

specified_name

Use this optional LONG VARCHAR parameter to specify the identifier for the object indicated in `specified_scope`. The default is NULL.

do_commit

Use this optional TINYINT parameter to specify whether to commit, automatically, rows inserted by this procedure. The default is 1. Specify 1 to commit the rows automatically (recommended), and 0 to not commit them automatically.

Remarks

This procedure replaces the rows in the sa_diagnostic_tracing_level table, changing the tracing level and scope to the settings specified when the procedure is called.

Setting the level 0 does not stop the tracing session. Instead, the tracing session remains attached to the tracing database, but no tracing data is sent. The tracing session is still active when the level is 0.

This system procedure must be called from the database being profiled.

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must have the DIAGNOSTICS system role, and the MANAGE PROFILING system privilege.

Side Effects

None

Example

The following example sets the tracing level to 1. The entire database is profiled for performance counter data, and some samples of executed statements:

```
CALL sa_set_tracing_level( 1 );
```

The following example sets the tracing level to 3, and specifies the user AG84756. Only activities associated with AG84756 are traced:

```
CALL sa_set_tracing_level( 3, 'user', 'AG84756' );
```

Related Information

Customized Diagnostic Tracing Levels (Deprecated)
Diagnostic Tracing Scopes (Deprecated)
Diagnostic Tracing (Deprecated)
sa_diagnostic_tracing_level Table (Deprecated) [page 1508]

1.6.8.90 sa_snapshots System Procedure

Returns a list of snapshots that are currently active.

Syntax

sa_snapshots( )

Result Set

Column name	Data type	Description
connection_num	INTEGER	The connection ID for the connection on which the snapshot is running.
start_sequence_num	UNSIGNED BIGINT	A unique number that identifies the snapshot.
statement_level	BIT	True if the snapshot was created with statement-snapshot or readonly-statement-snapshot. Otherwise, false.

Remarks

Several statement snapshots can exist on one connection. For nested or interleaved statements running under statement snapshot isolation levels, each one begins a different statement snapshot with its first read or update.

Usually there is only one transaction snapshot per connection (one entry per connection in sa_snapshots with statement_level=0). However, a snapshot associated with a cursor never changes after the cursor's first fetch

and a cursor opened WITH HOLD stays open through a commit or rollback. If the cursor has an associated snapshot, then the snapshot also persists. Therefore, it is possible for multiple transaction snapshots to exist for the same connection_num: one for the current transaction snapshot and one or more for old transaction snapshots that persist because of WITH HOLD cursors.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR system privilege.

> **i** Note
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

You can execute the following query to identify snapshots that are currently active.

```
CALL sa_snapshots( );
```

Related Information

Snapshot Isolation
sa_transactions System Procedure [page 1741]

1.6.8.91 sa_split_list System Procedure

Takes a string of values, separated by a delimiter, and returns a set of rows (one row for each value).

> Syntax
>
> *sa_split_list(*

```
str
 [, delim
 [, maxlen ] ]
)
```

Parameters

str

Use this LONG VARCHAR parameter to specify the string containing the values to be split, separated by `delim`.

delim

Use this optional CHAR(10) parameter to specify the delimiter used in `str` to separate values. The delimiter can be a string of any characters, up to 10 bytes. If `delim` is not specified, a comma is used by default.

maxlen

Use this optional INTEGER parameter to specify the maximum length of the returned values. For example, if `maxlen` is set to 3, the values in the result set are truncated to a length of 3 characters. If you specify 0 (the default), values can be any length.

Result Set

Column name	Data type	Description
line_num	INTEGER	Sequential number for the row.
row_value	LONG VARCHAR	Value from the string, truncated to `maxlen` if required.

Remarks

The sa_split_list procedure accepts a string with a delimited list of values, and returns a result set with one value per row. This is the opposite of the action performed by the LIST function [Aggregate]. An empty string is returned for row_value if the string:

- begins with `delim`
- contains two successive instances of `delim` in the middle of the string
- ends with `delim`

White space within the input string is significant. If the delimiter is a space character, extra spaces in the input string result in extra rows in the result set. If the delimiter is not a space character, spaces in the input string are not trimmed from the values in the result set.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following query returns a list of black colored products.

```
SELECT list( Name )
  FROM Products
  WHERE Color = 'Black';
```

list (Products.Name)
Tee Shirt,Baseball Cap,Visor,Shorts

In the following example, the sa_split_list procedure is used to return the original result set from the aggregated list.

```
SELECT *
  FROM sa_split_list( 'Tee Shirt,Baseball Cap,Visor,Shorts' );
```

line_num	row_value
1	Tee Shirt
2	Baseball Cap
3	Visor
4	Shorts

The following example returns a row for each word. To avoid returning rows where row_value is an empty string, the WHERE clause must be specified.

```
SELECT *
  FROM sa_split_list( 'one||three|four||six|', '|' )
  WHERE row_value <> '';
```

line_num	row_value
1	one
3	three
4	four

line_num	row_value
6	six

In the following example, a procedure called ProductsWithColor is created. When called, the ProductsWithColor procedure uses sa_split_list to parse the color values specified by the user, looks in the Color column of the Products table, and returns the name, description, size, and color for each product that matches one of the user-specified colors.

The result of the procedure call below is the name, description, size, and color of all products that are either white or black.

```
CREATE OR REPLACE PROCEDURE ProductsWithColor( IN color_list LONG VARCHAR )
BEGIN
  SELECT Name, Description, Size, Color
  FROM Products
  WHERE Color IN ( SELECT row_value FROM sa_split_list( color_list ) );
END;
SELECT * from ProductsWithColor( 'white,black' );
```

Related Information

LIST Function [Aggregate] [page 439]

1.6.8.92 sa_stack_trace System Procedure

Returns the stack trace leading to the current call location.

Syntax

```
sa_stack_trace(
[ stack_frames
[, detail_level
[, connection_id ] ] ]
)
```

Parameters

stack_frames

Use this optional CHAR(128) parameter to specify one of the following:

'procedure'

Return procedures but not the outer-most statement. This is the default behavior.

'caller'

Return only the outer-most statement (the statement that arrived from the client).

'procedure+caller', 'caller+procedure'

Return all statements.

detail_level

Use this optional CHAR(128) parameter to specify one of the following:

'stack'

Include procedure names and line numbers. This is the default behavior.

'stack+sql', 'sql+stack'

Include the procedure names and line numbers, as well as the SQL text of the statement being executed at each level.

connection_id

Use this optional UNSIGNED INTEGER parameter to filter the results returned to the specified connection ID. If not specified, information for the current connection is returned.

Result Set

Column name	Data type	Description
StackLevel	UNSIGNED SMALLINT	The current line has Stack Level 1.
UserName	CHAR(128)	The name of the owner of the procedure or trigger, or NULL if the current level is in a batch.
ProcName	CHAR(128)	The name of the procedure or trigger where the call was performed or the batch type.
LineNumber	UNSIGNED INTEGER	The line number of the call within the procedure, trigger, or batch.
SQLStatement	LONG VARCHAR	The statement being executed.

Remarks

Each record in the result set represents a single call on the stack. If the compound statement is not part of a procedure, function, trigger, or event, then the type of batch (watcom_batch or tsql_batch) is returned instead of the procedure name.

This function returns line numbers as found in the proc_defn column of the SYSPROCEDURE system table for the procedure. These line numbers might differ from those of the source definition used to create the procedure.

This procedure returns the same information as the STACK_TRACE function.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None.

Example

This example shows how to obtain the result set columns from the sa_stack_trace system procedure:

```
SELECT StackLevel, UserName, ProcName, LineNumber FROM sa_stack_trace();
```

When this statement is executed outside of the context of a stored procedure, the result set is empty.

The following example shows the implementation of a general stack trace procedure that sends its results to the client window:

```
CREATE OR REPLACE PROCEDURE StackDump( MSG CHAR(128) )
BEGIN
    DECLARE myStackLevel UNSIGNED SMALLINT;
    DECLARE myUserName CHAR(128);
    DECLARE myProcName CHAR(128);
    DECLARE myLineNumber UNSIGNED SMALLINT;
    DECLARE mySQLStatement long varchar;
    DECLARE err_notfound
        EXCEPTION FOR SQLSTATE '02000';
    DECLARE myStack CURSOR FOR
        SELECT StackLevel, UserName, ProcName, LineNumber, SQLStatement FROM
sa_stack_trace('caller+procedure', 'stack+sql');
    MESSAGE 'Stack Trace: ' || MSG TO CLIENT;
    OPEN myStack;
    StackLoop: LOOP
        FETCH NEXT myStack
            INTO myStackLevel, myUserName, myProcName, myLineNumber,
mySQLStatement;
        IF SQLSTATE = err_notfound THEN
            LEAVE StackLoop;
        END IF;
        IF myStackLevel != 1 THEN
            MESSAGE myStackLevel - 1 || ' ' || myUserName || ' ' || myProcName
|| ' ' || myLineNumber || ' ' || mySQLStatement
                TO CLIENT;
        ENDIF
    END LOOP StackLoop;
    CLOSE myStack;
END;

CREATE OR REPLACE PROCEDURE Proc1()
BEGIN
    CALL Proc2();
END;

CREATE OR REPLACE PROCEDURE Proc2()
```

```
BEGIN
    CALL Proc3();
END;

CREATE OR REPLACE PROCEDURE Proc3()
BEGIN
    CALL StackDump('Snapshot from Proc3');
END;

CALL Proc1();
```

Results:

```
CALL Proc1();
-- Stack Trace: Snapshot from Proc3
-- 1 DBA proc3 3 call StackDump('Snapshot from Proc3')
-- 2 DBA proc2 3 call Proc3()
-- 3 DBA proc1 3 call Proc2()
-- Procedure completed
```

Related Information

Exception Handling and Nested Compound Statements
TRY Statement [page 1446]
BEGIN Statement [page 784]
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_MESSAGE Function [Miscellaneous] [page 356]
ERROR_PROCEDURE Function [Miscellaneous] [page 358]
ERROR_SQLCODE Function [Miscellaneous] [page 359]
ERROR_SQLSTATE Function [Miscellaneous] [page 361]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
STACK_TRACE Function [Miscellaneous] [page 566]
sa_error_stack_trace System Procedure [page 1589]
SYSPROCEDURE System View [page 1931]

1.6.8.93 sa_statement_text System Procedure

Formats a SELECT statement so that individual items appear on separate lines. This is useful when viewing long statements from the request log, in which all newline characters are removed.

Syntax

```
sa_statement_text( txt )
```

Parameters

txt

Use this LONG VARCHAR parameter to specify a SELECT statement.

Result Set

Column name	Data type	Description
stmt_text	LONG VARCHAR	A clause of the SELECT statement.

Remarks

The text that is entered must be a string (in single quotes) or a string expression.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following call formats a SELECT statement so that individual items appear on separate lines.

```
CALL sa_statement_text( 'SELECT * FROM car WHERE name=''Audi''' );
```

	stmt_text
1	`SELECT *`
2	`FROM car`
3	`WHERE name = 'Audi'`

Related Information

sa_get_request_times System Procedure [page 1606]
sa_get_request_profile System Procedure [page 1604]

1.6.8.94 sa_table_fragmentation System Procedure

Reports information about the fragmentation of database tables.

Syntax

```
sa_table_fragmentation(
[ tbl_name
[, owner_name ] ]
)
```

Parameters

tbl_name

Use this optional CHAR(128) parameter to specify the name of the table to check for fragmentation. The default is NULL.

owner_name

Use this optional CHAR(128) parameter to specify the owner of `tbl_name`. The default is NULL.

Result Set

Column name	Data type	Description
TableName	CHAR(128)	Name of the table.
rows	UNSIGNED INTEGER	Number of rows in the table.
row_segments	UNSIGNED BIGINT	Number of row segments in the table.
segs_per_row	DOUBLE	Number of segments per row.

Remarks

Database administrators can use this procedure to obtain information about the fragmentation in a database's tables. If no arguments are supplied, results are returned for all tables in the database.

When database tables become excessively fragmented, you can run REORGANIZE TABLE or rebuild the database to reduce table fragmentation and improve performance.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY STATISTICS or MONITOR system privilege.

Side Effects

None

Example

```
CALL sa_table_fragmentation( 'Products','GROUPO' );
```

```
CALL sa_table_fragmentation( owner_name='GROUPO' );
```

Related Information

Database Rebuilds
REORGANIZE TABLE Statement [page 1336]

1.6.8.95 sa_table_page_usage System Procedure

Reports information about the page usage of database tables.

Syntax

sa_table_page_usage( )

Result Set

Column name	Data type	Description
TableId	UNSIGNED INTEGER	The table ID.
TablePages	INTEGER	The number of table pages used by the table.
PctUsedT	INTEGER	The percentage of used table page space.
IndexPages	INTEGER	The number of index pages used by the table.
PctUsedI	INTEGER	The percentage of used index page space.
PctOfFile	INTEGER	The percentage of the total database file the table occupies.
TableName	CHAR(128)	The table name.

Remarks

The results include the same information provided by the Information utility. When the progress_messages database option is set to Raw or Formatted, progress messages are sent from the database server to the client while the sa_table_page_usage system procedure is running.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY DBSPACE system privilege.

Side Effects

None

Example

The following example obtains information about the page usage of the SalesOrderItems table.

```
SELECT * FROM sa_table_page_usage( )
    WHERE TableName = 'SalesOrderItems';
```

Related Information

Information Utility (dbinfo)
progress_messages Option

1.6.8.96 sa_table_stats System Procedure

Reports information about how many pages have been read from each table.

Syntax

```
sa_table_stats( )
```

Result Set

Column name	Data type	Description
table_id	INTEGER	The table ID.
creator	CHAR(128)	The user name of the table's creator.
table_name	CHAR(128)	The table name.
"count"	UNSIGNED BIGINT	The estimated number of rows in the table, taken from SYSTAB.
table_page_count	UNSIGNED BIGINT	The number of main pages used by the table.
table_page_cached	UNSIGNED BIGINT	The number of base table pages currently stored in the cache. Index pages are not included.
table_page_reads	UNSIGNED BIGINT	The number of page reads performed for pages in the main table.
ext_page_count	UNSIGNED BIGINT	The total number of extension pages used by the underlying table. Extension pages store the ends of long strings or other BLOB types.
ext_page_cached	UNSIGNED BIGINT	The number of extension pages currently stored in the cache.
ext_page_reads	UNSIGNED BIGINT	The number of extension pages that have ever been read into cache.

Remarks

Each row returned by the sa_table_stats procedure describes a table for which the optimizer is maintaining page statistics. The sa_table_stats procedure can be used to find which tables are using cache memory and how many disk reads are being performed for each table. For example, you can use the sa_table_stats procedure to find the table that is generating the most disk reads. The results of the procedure represent estimates and should be used only for diagnostic purposes.

The table_page_cached column indicates how many pages of the table are currently stored in the cache, and the table_page_reads column indicates how many table pages have been read from disk since the optimizer started maintaining counts for the table. These statistics are not stored persistently within the database; they represent the activity on tables after they are loaded into memory for the first time.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR system privilege.

> **i Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

You can execute the following query to get information about how many pages have been read from each table.

```
CALL sa_table_stats( );
```

Related Information

SYSTAB System View [page 1950]

1.6.8.97 sa_text_index_stats System Procedure

Returns statistical information about the text indexes in the database.

Syntax

```
sa_text_index_stats( )
```

Remarks

Use the sa_text_index_stats system procedure to view statistical information for each text index in the database. The following table describes the information returned by sa_text_index_stats.

Column name	Type	Description
owner_id	UNSIGNED INTEGER	ID of the owner of the table.
table_id	UNSIGNED INTEGER	ID of the table.
index_id	UNSIGNED INTEGER	ID of the text index.
text_config_id	UNSIGNED BIGINT	ID of the text configuration object referenced by the index.
owner_name	CHAR(128)	Name of the owner.
table_name	CHAR(128)	Name of the table.
index_name	CHAR(128)	Name of the text index.
text_config_name	CHAR(128)	Name of the text configuration object.
doc_count	UNSIGNED BIGINT	Total number of indexed column values in the text index.
doc_length	UNSIGNED BIGINT	Total length of data in the text index.
pending_length	UNSIGNED BIGINT	Total length of the pending changes.
deleted_length	UNSIGNED BIGINT	Total length of the pending deletions.
last_refresh	TIMESTAMP	Database server local date and time of the last refresh. This value is affected by simulated time zone.

The pending_length, deleted_length, and last_refresh values are 0 for IMMEDIATE REFRESH text indexes.

For MANUAL REFRESH text indexes, you can use doc_length, pending_length, and deleted_length to decide whether to refresh the text index, and the type of refresh to perform (rebuild vs. incremental).

Privileges

You must have EXECUTE privilege on the system procedure, as well as one of the following system privileges:

- MANAGE ANY STATISTICS
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX
- CREATE ANY OBJECT
- ALTER ANY OBJECT
- DROP ANY OBJECT

Side Effects

None

Example

The following statement returns statistical information for each text index in the database:

```
CALL sa_text_index_stats( );
```

Related Information

Full Text Search
Text Index Concepts and Reference
DROP TEXT INDEX Statement [page 1136]
REFRESH TEXT INDEX Statement [page 1325]
TRUNCATE TEXT INDEX Statement [page 1445]
sa_refresh_text_indexes System Procedure [page 1683]
sa_text_index_vocab System Procedure [page 1737]
SYSTEXTIDX System View [page 1960]

1.6.8.98 sa_text_index_vocab System Procedure

Lists all terms that appear in a CHAR text index, and the total number of indexed values that each term appears in.

Syntax

```
sa_text_index_vocab(
indexname
, tabname
[, tabowner ]
)
```

Parameters

indexname

Use this CHAR(128) parameter to specify the name of the text index.

tabname

Use this CHAR(128) parameter to specify the name of the table on which the text index is built.

tabowner

Use this optional CHAR(128) parameter to specify the owner of the table. The default is NULL.

Result Set

Column name	Data type	Description
term	VARCHAR(60)	A term in the text index.
freq	BIGINT	The number of indexed values the term appears in.

Remarks

The sa_text_index_vocab system procedure returns all terms that appear in a text index, and the total number of indexed values that each term appears in (which is less than the total number of occurrences if the term appears multiple times in some indexed values).

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must have the SELECT object-level privilege on the indexed table, or have the SELECT ANY TABLE system privilege.

Side Effects

None

Example

The following example builds a text index called VocabTxtIdx on the Products.Description column in the sample database.

```
CREATE TEXT INDEX VocabTxtIdx2 ON Products( Description );
```

The next statement executes the sa_text_index_vocab system procedure to return all the terms that appear in the text index.

```
SELECT * FROM sa_text_index_vocab( 'VocabTxtIdx2', 'Products', 'GROUPO' );
```

term	freq
Cap	2
Cloth	1
Cotton	2
Crew	1
Hooded	1
neck	2
...	...

Related Information

Full Text Search
Full Text Term and Phrase Searches
sa_text_index_vocab_nchar System Procedure [page 1739]
DROP TEXT INDEX Statement [page 1136]
REFRESH TEXT INDEX Statement [page 1325]
TRUNCATE TEXT INDEX Statement [page 1445]
sa_refresh_text_indexes System Procedure [page 1683]
SYSTEXTIDX System View [page 1960]

1.6.8.99 sa_text_index_vocab_nchar System Procedure

Lists all terms that appear in an NCHAR text index, and the total number of indexed values that each term appears in.

Syntax

```
sa_text_index_vocab_nchar(
indexname
, tabname
[, tabowner ]
)
```

Parameters

indexname

Use this CHAR(128) parameter to specify the name of the text index.

tabname

Use this CHAR(128) parameter to specify the name of the table on which the text index is built.

tabowner

Use this optional CHAR(128) parameter to specify the owner of the table. The default is NULL.

Result Set

Column name	Data type	Description
term	NVARCHAR(60)	A term in the text index.
freq	BIGINT	The number of indexed values the term appears in.

Remarks

The sa_text_index_vocab_nchar system procedure returns all terms that appear in a text index, and the total number of indexed values that each term appears in (which is less than the total number of occurrences if the term appears multiple times in some indexed values).

Privileges

You must have EXECUTE privilege on the system procedure.

Additionally, you must have the SELECT object-level privilege on the indexed table, or have the SELECT ANY TABLE system privilege.

Side Effects

None

Example

The following example builds a text index called VocabNTxtIdx on the NProducts.Description column. The NProducts table is a version of the Products table with NCHAR columns instead of CHAR columns.

```
CREATE TEXT INDEX VocabNTxtIdx2 ON NProducts( Description );
```

The next statement executes the sa_text_index_vocab_nchar system procedure to return all the terms that appear in the text index.

```
SELECT * FROM sa_text_index_vocab_nchar( 'VocabNTxtIdx2', 'NProducts' );
```

term	freq
Cap	2
Cloth	1
Cotton	2
Crew	1
Hooded	1
neck	2
...	...

Related Information

Full Text Search
Full Text Term and Phrase Searches
sa_text_index_vocab System Procedure [page 1737]
DROP TEXT INDEX Statement [page 1136]
REFRESH TEXT INDEX Statement [page 1325]
TRUNCATE TEXT INDEX Statement [page 1445]
sa_refresh_text_indexes System Procedure [page 1683]
SYSTEXTIDX System View [page 1960]

1.6.8.100 sa_transactions System Procedure

Returns a list of transactions that are currently active.

Syntax

```
sa_transactions( )
```

Result Set

Column name	Data type	Description
connection_num	INTEGER	The connection ID for the connection the transaction is running on.
transaction_id	INTEGER	The ID that uniquely identifies the transaction as long as the database server keeps track of it. IDs are reused as old transaction information is discarded.
start_time	TIMESTAMP	The database server local date and time when the transaction started. This value is affected by simulated time zone.
start_sequence_num	UNSIGNED BIGINT	The start sequence number for the transaction.
end_sequence_num	UNSIGNED BIGINT	Then end sequence number for the transaction if it has been committed or rolled back, otherwise, NULL.
committed	BIT	The state of the transaction: true if the transaction ended with a COMMIT, false if it ended with a ROLLBACK, and NULL if the transaction is still active.
version_entries	UNSIGNED INTEGER	The count of the number of row versions the transaction has saved.

Remarks

This procedure provides information about the transactions that are currently running against the database.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MONITOR system privilege.

> **Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Side Effects

None

Example

You can execute the following query to identify transactions that are currently active.

```
CALL sa_transactions( );
```

Related Information

Snapshot Isolation
sa_snapshots System Procedure [page 1721]

1.6.8.101 sa_unload_cost_model System Procedure

Unloads the current cost model to the specified file.

> **Syntax**
>
> *sa_unload_cost_model(file_name)*

Parameters

file_name

Use this CHAR(256) parameter to specify the name of the file in which to unload the data. Because it is the database server that executes the system procedure, `file_name` specifies a file on the database server computer, and a relative `file_name` specifies a file relative to the database server's starting directory.

Remarks

The optimizer uses cost models to determine optimal access plans for queries. The database server maintains a cost model for each database. The cost model for a database can be recalibrated at any time using the CALIBRATE SERVER clause of the ALTER DATABASE statement. For example, you might decide to recalibrate the cost model if you move the database onto non-standard hardware.

The sa_unload_cost_model system procedure allows you save a cost model to an ASCII file (`file_name`). You can then log into another database and use the sa_load_cost_model system procedure to load the cost model from the first database into the second one. This avoids having to recalibrate the second database.

> **i** Note
>
> The sa_unload_cost_model system procedure does not include CALIBRATE PARALLEL READ information in the file.

Using the sa_unload_cost_model system procedure eliminates repetitive, time-consuming recalibration activities when there is a large number of similar hardware installations.

You must have write permissions where the file is created.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SELECT ANY TABLE system privilege.

Side Effects

None

Example

The following example unloads the cost model to a file called `costmodel8`:

```
CALL sa_unload_cost_model( 'costmodel8' );
```

Related Information

Advanced: Query Optimization
ALTER DATABASE Statement [page 662]
sa_load_cost_model System Procedure [page 1632]

1.6.8.102 sa_user_defined_counter_add System Procedure

Adjusts the value of a user-defined counter by a specified amount.

Syntax

```
sa_user_defined_counter_add(
counter_name
[, delta
[, apply_to_con
[, appy_to_db
[, apply_to_server ] ] ] ]
)
```

Parameters

counter_name

Use this VARCHAR(128) parameter to specify the name of the user-defined counter whose value you want to change. Examples of user-defined counter names are UserDefinedCounterRate01 and UserDefinedCounterRaw01.

delta

Use this BIGINT parameter to specify the amount that the user-defined counter is incremented or decremented by. The default is 1.

apply_to_con

Use this INTEGER parameter to specify whether to adjust the counter value for the current connection. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

apply_to_db

Use this INTEGER parameter to specify whether to adjust the counter value for the database. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

apply_to_server

Use this INT parameter to specify whether to adjust the counter value for the database server. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

Returns

This function returns an INTEGER status code.

Remarks

This function returns 1 if `delta` is defined, 0 if `delta` is not defined, and an error code if an error occurs. Examples of errors include:

- an invalid counter name
- an invalid value for the apply_to_server, apply_to_db, or apply_to_con parameter.

Concurrent access to counters is applied atomically, so a counter value can be incremented from multiple, concurrent requests.

User-defined counters are implemented as 32-bit UNSIGNED INTEGER values.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

The following statement increments the value of UserDefinedCounterRate01 by 2 for the current connection, database, and database server:

```
SELECT sa_user_defined_counter_add( 'UserDefinedCounterRate01', 2, 1, 1, 1 );
```

Related Information

User-defined Properties
sa_user_defined_counter_set System Procedure [page 1746]
List of Connection Properties
List of Database Server Properties

1.6.8.103 sa_user_defined_counter_set System Procedure

Sets a user-defined counter to a specified value.

Syntax

```
sa_user_defined_counter_set(
counter_name
, value
[, apply_to_con
[, appy_to_db
[, apply_to_server ] ] ] )
```

Parameters

counter_name

Use this VARCHAR(128) parameter to specify the name of the user-defined counter whose value you want to change. Examples of user-defined counter names are UserDefinedCounterRate01 and UserDefinedCounterRaw01.

value

Use this BIGINT parameter to specify the value to which the user-defined counter is set.

apply_to_con

Use this INT parameter to specify whether to adjust the counter value for the current connection. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

apply_to_db

Use this INT parameter to specify whether to adjust the counter value for the database. 0 means do not adjust the value, and 1 means adjust the value. The default is 0.

apply_to_server

Use this INT parameter to specify whether to adjust the counter value for the database server. 0 means do not adjust the value, and 1 means adjust the value. The default is 0.

Returns

This function returns an INTEGER status code.

Remarks

This function returns 1 if `value` is defined, 0 if `value` is not defined, and an error code if an error occurs. Examples of errors include:

- an invalid counter name
- an invalid value for the apply_to_server, apply_to_db, or apply_to_con parameter.

Concurrent access to counters is applied atomically, so a counter value can be reset from multiple, concurrent requests.

User-defined counters are implemented as 32-bit UNSIGNED INTEGER values.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

The following statement sets the value of UserDefinedCounterRate01 to 0 for the current connection, database, and database server:

```
SELECT sa_user_defined_counter_set( 'UserDefinedCounterRate01', 0, 1, 1, 1 );
```

Related Information

User-defined Properties
sa_user_defined_counter_add System Procedure [page 1744]
List of Connection Properties
List of Database Server Properties

1.6.8.104 sa_validate System Procedure

Validates all or parts of a database.

Syntax

```
sa_validate(
[ tbl_name
[, owner_name ] ]
[, check_type ] ]
[, isolation_type ] ]
)
```

Parameters

tbl_name

Use this optional CHAR(128) parameter to specify the name of a table or materialized view to validate. The default is NULL, in which case the entire database is validated.

owner_name

Use this optional CHAR(128) parameter to specify an owner. When specified by itself, all tables and materialized views owned by the owner are validated. The default is NULL.

check_type

Use this optional CHAR(10) parameter to specify the type of validation to perform. The possible values are

EXPRESS

If this parameter is EXPRESS, each table is checked using a VALIDATE TABLE statement with the WITH EXPRESS CHECK clause.

NULL If this parameter is NULL (the default), each table is checked using a VALIDATE TABLE statement.

isolation_type

Use this optional parameter when validating tables that have active transactions to prevent receiving false errors about corrupt tables. The possible values are:

DATA LOCK Prevents transactions from modifying the table schema or data by applying exclusive data locks on the specified tables. Concurrent transactions can read, but not modify the table data or schema.

SNAPSHOT Ensures that only committed data is checked by applying snapshot isolation. Transactions can read and modify the data. This clause requires that the database have snapshot isolation enabled (with the allow_snapshot_isolation database option). Because this clause uses snapshot isolation, performance is often affected.

Result Set

Column name	Data type	Description
Messages	CHAR(128)	If validation succeeds without error, then the column contains `No error detected`.
IsValid	BIT	1 if valid; 0 if validation errors are detected
ObjectName	CHAR(261)	• Empty string if valid • Database if database validation fails • The owner and name of a table if table validation fails.

Remarks

Argument specified	Type of validation
None	All tables, materialized views, and indexes in the database are validated (equivalent to a VALIDATE TABLE on every table). The database itself is also validated (equivalent to a VALIDATE DATABASE statement), including checksum validation.
`tbl_name`	If `owner_name` is NULL, then all tables matching `tbl_name`, their materialized views, and their indexes are validated.
`owner_name`	All tables, materialized views, and indexes owned by the specified user are validated.
`tbl_name` and `owner_name`	The specified table or materialized view owned by the specified user, and all of its indexes, are validated.
`check_type`	The specified table(s), materialized view(s), or index(es) is validated using WITH EXPRESS CHECK. The database itself is also validated, including checksum validation.
`isolation_type`	The specified table(s), materialized view(s), or index(es) is validated. Tables either have exclusive data locks applied to them, or only committed data is evaluated. The database itself is also validated, including checksum validation.

> ⚠ Caution
>
> If `isolation_type` is not specified, then only perform validation while no connections are making changes to the database; otherwise, false errors may be reported indicating some form of database corruption.

You can validate disk pages for databases that use global or write checksums. If a database has global checksums enabled, then all database pages are validated. If a database has used only write checksums, then only pages with checksums are validated.

For databases with checksums enabled, a checksum is calculated for each database page and this value is stored when the page is written to disk. You can use the Validation utility (dbvalid), the VALIDATE statement, the sa_validate system procedure, or the Validate Database Wizard in SQL Central to perform checksum validation, which consists of reading the database pages from disk and calculating the checksum for the page. If the calculated checksum does not match the stored checksum for a page, the page has been modified or corrupted while on disk or while writing to the page. If one or more pages has been corrupted, an error is returned and information about the invalid pages appears in the database server messages window.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the VALIDATE ANY OBJECT system privilege.

Side Effects

If `isolation type` is DATA LOCK, then exclusive data locks are applied to the specified table(s) or view(s).

Automatic commit for both `isolation_type` options.

Example

The following statement performs a validation of tables and materialized views owned by user pjones:

```
CALL sa_validate( owner_name = 'pjones' );
```

Related Information

Corruption Detection Using Checksums
VALIDATE Statement [page 1476]
Validation Utility (dbvalid)

1.6.8.105 sa_verify_password System Procedure

Validates the password of the current user.

Syntax

```
sa_verify_password( curr_pswd )
```

Parameters

curr_pswd

Use this CHAR(128) parameter to specify the password of the current database user.

Returns

The function returns an INTEGER value.

Remarks

This procedure is used by sp_password. If the password matches, 0 is returned and no error occurs. If the password does not match, an error is diagnosed. The connection is not terminated if the password does not match.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example attempts to validate the current connection's password when the current user is DBA or User1. An error occurs if the current password does not match.

```
IF USER_NAME() = 'DBA' THEN
    SELECT sa_verify_password( 'sql' );
ELSEIF USER_NAME() = 'User1' THEN
    SELECT sa_verify_password( 'user' );
END IF;
```

Related Information

Adaptive Server Enterprise System Procedures [page 1518]

1.6.8.106 sp_alter_secure_feature_key System Procedure

Alters a previously defined secured feature key by modifying the authorization code and/or feature list.

Syntax

```
sp_alter_secure_feature_key(
name
, auth_key
, features
)
```

Parameters

name

The VARCHAR (128) name for the secured feature key you want to alter. A key with the given name must already exist.

auth_key

The CHAR (128) case-sensitive new authorization code for the secured feature key. The authorization code must be either a non-empty string of at least six characters, or NULL, indicating that the existing authorization code is not to be changed.

features

The LONG VARCHAR, comma-separated list of features that the key can enable. If `features` is NULL, the existing feature set is not changed.

Remarks

You must have the MANAGE_KEYS feature enabled on the connection to run this procedure.

This procedure allows you to alter the authorization code or feature list of an existing secured feature key.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

In order for the following example to work, the server must be started with the option: `-sk securefkey`.

This example enables the SYSTEM secured feature key which includes MANAGE_KEYS, creates a new secured feature key called MYSET with case-sensitive authorization code SecureMySet, alters which secured features belong to the set, and then uses sp_list_secure_feature_keys to obtain a list of the currently defined secured feature keys:

```
CALL dbo.sp_use_secure_feature_key( 'system', 'securefkey' );
CALL dbo.sp_create_secure_feature_key( 'myset', 'SecureMySet', 'local' );
CALL dbo.sp_alter_secure_feature_key( 'myset', NULL, 'local,remote' );
CALL dbo.sp_list_secure_feature_keys( );
```

Related Information

Creating Secured Feature Keys
sp_create_secure_feature_key System Procedure [page 1760]
sp_drop_secure_feature_key System Procedure [page 1771]
sp_list_secure_feature_keys System Procedure [page 1791]
sp_use_secure_feature_key System Procedure [page 1854]
-sf Database Server Option

1.6.8.107 sp_auth_sys_role_info System Procedure

Returns the mapping of authorities from previous versions of SQL Anywhere to their corresponding compatibility roles.

Syntax

```
sp_auth_sys_role_info( )
```

Result Set

Column	Type	Description
auth	VARCHAR(20)	The name of the authority.
role_name	CHAR(128)	The name of the corresponding compatibility role.
role_id	UNSIGNED INTEGER	The ID number for the compatibility role.

Remarks

None

Privileges

You must have EXECUTE privilege on the system procedure.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the list of authorities (auth) from pre-16.0 SQL Anywhere versions, mapped to their corresponding compatibility roles (role_name) provided as of version 16.0.

```
CALL dbo.sp_auth_sys_role_info();
```

auth	role_name	role_id
DBA	SYS_AUTH_DBA_ROLE	2147485648
RESOURCE	SYS_AUTH_RESOURCE_ROLE	2147485649
BACKUP	SYS_AUTH_BACKUP_ROLE	2147485650
VALIDATE	SYS_AUTH_VALIDATE_ROLE	2147485651
READFILE	SYS_AUTH_READFILE_ROLE	2147485652
PROFILE	SYS_AUTH_PROFILE_ROLE	2147485653
READCLIENTFILE	SYS_AUTH_READCLIENTFILE_ROLE	2147485654
WRITECLIENTFILE	SYS_AUTH_WRITECLIENTFILE_ROLE	2147485655
WRITEFILE	SYS_AUTH_WRITEFILE_ROLE	2147485656
REMOTE DBA	SYS_RUN_REPLICATION_ROLE	2147485664

Related Information

Compatibility Roles
Authorities Become Compatibility Roles

1.6.8.108 sp_copy_directory System Procedure

Copies a directory to a specified location.

Syntax

```
sp_copy_directory(
source_path
, destination_path
)
```

Parameters

source_path

Use this LONG NVARCHAR parameter to specify the path of the directory to copy. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

destination_path

Use this LONG NVARCHAR parameter to specify the path where the directory is copied to. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located. The directory is created if it does not already exist

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function copies the directory and its files from the source directory to the specified directory. The directory and its files remain in the source directory. Use the sp_delete_directory system procedure to delete the source directory.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE and WRITE FILE system privileges.

Example

The following statement makes a copy of the subdirectories and files in the `SQLAnywhere` directory in `SQLAnywhere.bkp`.

```
SELECT dbo.sp_copy_directory('c:\\sqlany\\samples\\SQLAnywhere', 'c:\\sqlany\
\samples\\SQLAnywhere.bkp');
```

The entire directory including its subdirectories and files are duplicated.

Related Information

sp_copy_file System Procedure [page 1757]

sp_create_directory System Procedure [page 1758]
sp_delete_directory System Procedure [page 1764]
sp_delete_file System Procedure [page 1765]
sp_list_directory System Procedure [page 1786]
sp_move_directory System Procedure [page 1793]
sp_move_file System Procedure [page 1795]
Directory and File System Procedures [page 1516]

1.6.8.109 sp_copy_file System Procedure

Copies a file to a specified location.

Syntax

```
sp_copy_file(
source_path
, destination_path
)
```

Parameters

source_path

Use this LONG NVARCHAR parameter to specify the file path, including file name, of the file to move. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, then absolute and relative paths refer to the directory where the main database file is located.

destination_path

Use this LONG NVARCHAR parameter to specify the new directory of the file. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function copies a file from one directory to another. The file remains in the source directory. You can delete the file in the source directory using the sp_delete_file system procedure.

If disk sandboxing is enabled, the source and destination paths must be accessible.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE and WRITE FILE system privileges.

Example

The following statement copies the file `license.txt` to the `c:\temp` directory and gives it a new name.

```
SELECT dbo.sp_copy_file('c:\\sqlany\\license.txt', 'c:\\temp\\license.bkp');
```

A duplicate copy of the `license.txt` file exists in the `temp` directory under a different name.

Related Information

Disk Sandboxing
sp_copy_directory System Procedure [page 1755]
sp_create_directory System Procedure [page 1758]
sp_delete_directory System Procedure [page 1764]
sp_delete_file System Procedure [page 1765]
sp_list_directory System Procedure [page 1786]
sp_move_directory System Procedure [page 1793]
sp_move_file System Procedure [page 1795]
Directory and File System Procedures [page 1516]

1.6.8.110 sp_create_directory System Procedure

Creates a directory on the computer.

Syntax

```
sp_create_directory( root_path )
```

Parameters

root_path

Use this LONG NVARCHAR parameter to specify the directory path to create. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function creates a new directory.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the WRITE FILE system privilege.

Example

The following statement creates a directory named `SQLAnywhere.bkp` in the `c:\sqlany\samples` directory.

```
SELECT dbo.sp_create_directory( 'c:\\sqlany\\samples\\SQLAnywhere.bkp' );
```

Related Information

Creating Secured Feature Keys
sp_copy_directory System Procedure [page 1755]
sp_copy_file System Procedure [page 1757]
sp_delete_directory System Procedure [page 1764]
sp_delete_file System Procedure [page 1765]
sp_list_directory System Procedure [page 1786]
sp_move_directory System Procedure [page 1793]

sp_move_file System Procedure [page 1795]
Directory and File System Procedures [page 1516]

1.6.8.111 sp_create_secure_feature_key System Procedure

Creates a new secured feature key.

Syntax

```
sp_create_secure_feature_key(
name
, auth_key
, features
)
```

Parameters

name

The VARCHAR (128) name for the new secured feature key. This argument cannot be NULL or an empty string.

auth_key

The CHAR (128) case-sensitive authorization code for the secured feature key. The authorization code must be a non-empty string of at least six characters.

features

The LONG VARCHAR comma-separated list of features that the new key can enable.

Specifying **–** before a feature means that the feature is not re-enabled when the secured feature key is acquired.

Remarks

You must have the MANAGE_KEYS feature enabled on the connection to run this system procedure.

This procedure creates a new secured feature key that can be given to any user. The SYSTEM secured feature key is created by using the -sk database server option.

There is a limit of 1000 secured feature keys per database server.

The authorization code must be a non-empty string of at least six characters, and it cannot contain double quotes, control characters (any character less than 0x20), or backslashes.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

In order for the following example to work, the server must be started with the option: `-sk securefkey`.

This example enables the SYSTEM secured feature key which includes MANAGE_KEYS, creates a new secured feature key called MYSET with a case-sensitive authorization code SecureMySet, and then uses sp_list_secure_feature_keys to obtain a list of the currently defined secured feature keys:

```
CALL dbo.sp_use_secure_feature_key( 'system', 'securefkey' );
CALL dbo.sp_create_secure_feature_key( 'myset', 'SecureMySet', 'local' );
CALL dbo.sp_list_secure_feature_keys( );
```

This example enables the SYSTEM secured feature key which includes MANAGE_KEYS and then creates a new secured feature key called MYSET2 with a case-sensitive authorization code SecureMySet2. The `database` feature is excluded from the feature set:

```
CALL dbo.sp_use_secure_feature_key( 'system', 'securefkey' );
CALL dbo.sp_create_secure_feature_key( 'myset2', 'SecureMySet2', 'local,-
database' );
```

Related Information

Creating Secured Feature Keys
-sf Database Server Option
-sk Database Server Option
sp_alter_secure_feature_key System Procedure [page 1752]
sp_drop_secure_feature_key System Procedure [page 1771]
sp_list_secure_feature_keys System Procedure [page 1791]
sp_use_secure_feature_key System Procedure [page 1854]

1.6.8.112 sp_db_cache_contents System Procedure

Returns one row per dbspace with a bitmap showing which pages are currently in the cache.

> **Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

> **Syntax**

```
sp_db_cache_contents( [ dbspace_id ] )
```

Parameters

dbspace_id Use this SMALLINT parameter to specify a single dbspace. If `dbspace_id` is NULL (default), then one row is returned for every dbspace belonging to the current database. Otherwise, one row is returned for the specified dbspace.

Remarks

This procedure returns a result set with one row per dbspace that has a LONG VARBIT object with 1 bit for each page of the dbspace that is currently in the cache.

Privileges

You must have the EXECUTE privilege on the system procedure. You must have the SERVER OPERATOR system privilege or the MONITOR system privilege.

Side Effects

None

Example

1. Create a dbspace by executing the following statement:

```
CREATE DBSPACE mydbs
AS 'C:\\mydb\\mydbs.db';
```

2. Create a table in the new dbspace and add some data by executing the following statements:

```
CREATE TABLE mytable( col1 INT, col2 CHAR(128) ) IN mydbs;
```

```
INSERT INTO mytable
    SELECT column_id, column_name
    FROM sys.syscolumn;
```

```
COMMIT;
```

3. Execute the following statement to determine the ID of the new dbspace:

```
SELECT * FROM SYS.SYSDBSPACE WHERE dbspace_name = 'mydbs';
```

   Note the ID of the new dbspace and that the saved_cache_pages column is NULL.

4. Verify which pages from the new dbspace are currently in the cache by executing the following statement:

```
SELECT * FROM dbo.sp_db_cache_contents( );
```

   The statement above returns information for all dbspaces in the database. To restrict the results to the new mydbs dbspace, execute the following statement, where `mydbs-ID` is the ID of the new mydbs dbspace that was obtained from SYS. SYSDBSPACE:

```
SELECT * FROM dbo.sp_db_cache_contents( mydbs-ID );
```

5. Save the current steady state of the database by executing the following statement:

```
ALTER DATABASE SAVE CACHE;
```

6. Later, after numerous other transactions have been executed and the cache has undergone significant change, you want to return to the saved steady state. You can either read all the steady state pages for all dbspaces into the cache or read all the steady state pages for a specific dbspace into the cache.
   To read all the steady state pages for all dbspaces into the cache, execute the following statement:

```
ALTER DATABASE RESTORE CACHE;
```

   If you would rather read all the steady state pages for one dbspace (for example, the new dbspace mydbs), then execute a series of statements similar to the following:

```
CREATE VARIABLE cache_pages LONG VARBIT;
```

```
SELECT saved_cache_pages
    INTO cache_pages
    FROM SYS.SYSDBSPACE
    WHERE dbspace_name='mydbs';
```

```
CALL dbo.sp_read_db_pages( mydbs-ID, cache_pages );
```

Related Information

Cache Warming
sp_read_db_pages System Procedure [page 1810]

1.6.8.113 sp_delete_directory System Procedure

Deletes the specified directory.

Syntax

```
sp_delete_directory( root_path )
```

Parameters

root_path

Use this LONG NVARCHAR parameter to specify the directory path of the directory to be deleted. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function deletes a directory from a specified location.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE and WRITE FILE system privileges.

Example

The following statement deletes the directory `SQLAnywhere.bkp`.

```
SELECT dbo.sp_delete_directory('c:\\sqlany\\samples\\SQLAnywhere.bkp');
```

Related Information

Creating Secured Feature Keys
sp_copy_directory System Procedure [page 1755]
sp_copy_file System Procedure [page 1757]
sp_create_directory System Procedure [page 1758]
sp_delete_file System Procedure [page 1765]
sp_list_directory System Procedure [page 1786]
sp_move_directory System Procedure [page 1793]
sp_move_file System Procedure [page 1795]
Directory and File System Procedures [page 1516]

1.6.8.114 sp_delete_file System Procedure

Deletes the specified file from the computer.

Syntax

```
sp_delete_file( file_path )
```

Parameters

file_path

Use this LONG NVARCHAR parameter to specify the file path, including file name, of the file to be deleted. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function deletes the specified file from the computer.

You must be connected to a database server on the computer where the file resides to run this procedure.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE and WRITE FILE system privileges.

Example

The following statement deletes the `license.bkp` file from the `temp` directory.

```
SELECT dbo.sp_delete_file('c:\\temp\\license.bkp');
```

Related Information

Creating Secured Feature Keys
sp_copy_directory System Procedure [page 1755]
sp_copy_file System Procedure [page 1757]
sp_create_directory System Procedure [page 1758]
sp_delete_directory System Procedure [page 1764]
sp_list_directory System Procedure [page 1786]
sp_move_directory System Procedure [page 1793]
sp_move_file System Procedure [page 1795]
Directory and File System Procedures [page 1516]

1.6.8.115 sp_disk_info System Procedure

Returns disk drive information for a given path.

Syntax

```
sp_disk_info( path )
```

Parameters

path

Use this LONG VARCHAR parameter to specify the file or directory path, including the name, to return disk drive information for. The path can be absolute or relative. A relative path is resolved relative to the location of the current database. The default for this parameter is NULL.

Result Set

Column name	Data type	Description
free_space	UNSIGNED BIGINT	The amount of free disk space in bytes.
total_space	UNSIGNED BIGINT	The total amount of disk space in bytes.

Remarks

If you do not specify the path parameter, you specify NULL, or you specify an empty string, then the sp_disk_info system procedure returns the disk information for the current database's location.

To prevent users from accessing disk information, enable the sp_disk_info secure feature.

Privileges

You must have the ACCESS DISK INFORMATION system privilege.

Side Effects

None

Example

Execute the following statement to return the free disk space and total disk space in bytes available for the file `myfile.txt`:

```
SELECT * FROM dbo.sp_disk_info( 'myfile.txt' );
```

Results similar to the following are returned:

free_space	total_space
83388907520	242042793984

Related Information

Disk Sandboxing
Directory and File System Procedures [page 1516]

1.6.8.116 sp_displayroles System Procedure

Returns all roles granted to the specified system privilege, system role, user-defined role, or user name, or displays the entire hierarchy tree of roles.

Syntax

```
sp_displayroles(
user_role_name
, display_mode
, grant_type
)
```

Parameters

user_role_name

Use this CHAR(128) parameter to specify the name of a system privilege, system role, user-defined role, or user name. If it is not specified or is NULL, then the current user is used by default.

display_mode

Use this VARCHAR(30) parameter to specify whether to return parent-level or child-level hierarchy, relative to `user_role_name`. If `display_mode` is not specified or is NULL, then only explicitly granted roles and privileges are returned (no inherited roles or privileges). Possible values for `display_mode` include the following:

expand_up

Shows the system roles granted to `user_role_name` in the parent hierarchy tree for `user_role_name`.

expand_down

Shows the system roles and privileges granted to `user_role_name`, including the role hierarchy tree for the child levels of `user_role_name`.

grant_type

Use this VARCHAR(30) parameter to control the grant type returned. If it is not specified, then ALL is used by default. Possible values for `grant_type` including the following:

no_admin

Shows all roles and system privileges granted to `user_role_name` with the WITH NO ADMIN OPTION or WITH ADMIN OPTION clause.

admin

Returns all roles and system privileges granted to `user_role_name` with the WITH ADMIN OPTION or WITH ADMIN ONLY OPTION clause.

all

Shows all roles/system privileges granted to `user_role_name`.

Result Set

Column name	Data type	Description
role_name	CHAR(128)	The role or system privilege granted to `user_role_name`.
parent_role_name	CHAR(128)	The role names for the parents of `user_role_name`.
grant_type	CHAR(10)	Information about whether `user_role_name` has administrative rights. Possible values: NO ADMIN, ADMIN, or ADMIN ONLY.
role_level	SMALLINT	With expand_down mode: Level is 1 for directly granted roles, 2 for the next level below, and so on. With expand_up mode: Level is 0 for the roles to which `user_role_name` has been granted, -1 for the next hierarchy above, and so on.

Remarks

For system privileges, the result shows the system privilege name instead of the system privilege role name. With expand_down mode, the parent_role_name is NULL for level 1 (directly granted roles). With the default mode, the role_level column is 1 and parent_role_name is NULL, since with default mode only the directly granted roles are displayed.

If this procedure is used for a user with mode expand_up, then no results are returned since a user resides at the top level in any role hierarchy. Similarly, if this procedure is used for an immutable system privilege, with mode expand_down, then no results are returned because an immutable system privilege resides at the

bottom level in any role hierarchy. The default mode is to display only the directly granted roles/system privileges.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Privileges

You must have EXECUTE privilege on the system procedure.

To return the system privileges or roles for another user ID or a role, you must also have the MANAGE ROLES system privilege.

Side Effects

None

Example

The following statement returns all roles granted to the user issuing the command.

```
CALL dbo.sp_displayroles();
```

This examples returns the list of system privileges granted to the SYS_SPATIAL_ADMIN_ROLE system role:

```
CALL dbo.sp_displayroles( 'SYS_SPATIAL_ADMIN_ROLE' );
```

role_name	parent_role_name	grant_type	role_level
PUBLIC	(NULL)	NO ADMIN	1
MANAGE ANY SPATIAL OBJECT	(NULL)	NO ADMIN	1

This examples returns the list of system privileges granted to the SYS_SPATIAL_ADMIN_ROLE, including all roles above it in the hierarchy of roles:

```
CALL sp_displayroles( 'SYS_SPATIAL_ADMIN_ROLE', 'expand_up');
```

role_name	parent_role_name	grant_type	role_level
SYS_AUTH_DBA_ROLE	dbo	ADMIN	-3
SYS_AUTH_DBA_ROLE	SYS_RUN_REPLICA-TION_ROLE	ADMIN	-3
SYS_AUTH_SSO_ROLE	SYS_AUTH_DBA_ROLE	ADMIN	-2
MANAGE ROLES	SYS_AUTH_SSO_ROLE	ADMIN	-1
MANAGE ROLES	SYS_REPLICATION_AD-MIN_ROLE	NO ADMIN	-1
SYS_SPATIAL_ADMIN_ROLE	MANAGE ROLES	ADMIN ONLY	0

The following statement returns all system privileges granted to the user User1:

```
CALL sp_displayroles( 'User1' );
```

Related Information

Roles
Viewing the Roles and Privileges for a User or Role (SQL)
Viewing the Roles and Privileges for a User or Role (SQL Central)
sp_sys_priv_role_info System Procedure [page 1836]
sp_has_role System Procedure [page 1782]
sp_proc_priv System Procedure [page 1805]
sp_objectpermission System Procedure [page 1796]

1.6.8.117 sp_drop_secure_feature_key System Procedure

Deletes a secured feature key.

Syntax

```
sp_drop_secure_feature_key( name )
```

Parameters

name

The VARCHAR (128) name of the secured feature key to drop.

Remarks

You must have the MANAGE_KEYS feature enabled on the connection to run this procedure.

If the named key does not exist, an error is returned. If the named key exists, it is deleted as long as it is not the last secured feature key that is allowed to manage features and secured feature keys. The SYSTEM secured feature key cannot be dropped unless there is another key that has the MANAGE_FEATURES and MANAGE_KEYS features enabled.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

In order for the following example to work, the server must be started with the option: `-sk securefkey`.

This example enables the SYSTEM secured feature key which includes MANAGE_KEYS and then drops the secured feature key called MYSET:

```
CALL dbo.sp_use_secure_feature_key( 'system', 'securefkey' );
CALL dbo.sp_drop_secure_feature_key( 'myset' );
```

Related Information

Creating Secured Feature Keys
sp_alter_secure_feature_key System Procedure [page 1752]
sp_create_secure_feature_key System Procedure [page 1760]
sp_list_secure_feature_keys System Procedure [page 1791]
sp_use_secure_feature_key System Procedure [page 1854]
-sf Database Server Option

1.6.8.118 sp_find_top_statements System Procedure

Reports performance statistics for each logged statement and plan combination.

Syntax

Note

Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Syntax

```
sp_find_top_statements( stmt_text, stmt_hash )
```

Parameters

stmt_text

(Optional) A LONG VARCHAR parameter that specifies a SQL statement string. The default is NULL.

stmt_hash

(Optional) An UNSIGNED BIGINT parameter that specifies a statement hash. The default is NULL.

Result Set

Column name	Data type	Description
stmt_hash	UNSIGNED BIGINT	Returns the statement identifier.
owner_name	CHAR(128)	Returns the name of the owner of the stored procedure. The value is NULL if the statement is not part of a stored procedure, user-defined function, trigger, or event. The value may also be NULL if the statement is executed as part of a stored procedure and the stored procedure is subsequently dropped. A statement executed as part of a stored procedure gets a hash that is related to the procedure, not the statement text.

Column name	Data type	Description
proc_name	CHAR(128)	Returns the name of the procedure that the statement belongs to. The value is NULL if the statement is not part of a stored procedure, user-defined function, trigger or event. The value may also be NULL if the statement is executed as part of a stored procedure and the stored procedure is subsequently dropped. A statement executed as part of a stored procedure gets a hash that is related to the procedure, not the statement text.
reusable_stmt_id	UNSIGNED INTEGER	Returns a unique identifier assigned to the statement within a procedure (not necessarily a line number). The value is NULL if the statement is not part of a stored procedure, user-defined function, trigger, or event.
plan_hash	UNSIGNED BIGINT	Returns the plan identifier.
max_seconds	DOUBLE	Returns the maximum runtime observed for the statement when executed with the current plan.
sum_runtime	DOUBLE	Returns the total runtime for the statement with the current plan.
sum_square_runtime	DOUBLE	Returns the sum of the squares of the observed runtimes. This value is used for the standard deviation calculation.
max_blocking_time	DOUBLE	Returns the maximum observed blocking time for the statement when executed with the current plan.
sum_blocking_time	DOUBLE	Returns the total blocking time for the statement executions using the current plan.
num_exec	UNSIGNED BIGINT	Returns the number of times the statement was executed using the current plan.
total_num_rows	UNSIGNED BIGINT	Returns the total number of rows returned or modified by the statement over all executions performed with the current plan.
last_max_time_utc	TIMESTAMP	Returns the date and time in Coordinated Universal Time (UTC) that the maximum runtime was last updated.
last_time_utc	TIMESTAMP	Returns the date and time in Coordinated Universal Time (UTC) that the statement statistics were last updated with the current plan.

Remarks

This procedure returns one or more rows for each logged statement, with each row indicating the execution plan that was used.

Specify the stmt_text parameter if you want to see the hash for the specified statement, as well as the logged results for the statement, if any. If there is no data for the hash, a single row with hash and NULL is returned. If you want to fetch data for the statement and know the hash, specify the stmt_hash parameter. Otherwise, do not specify either parameter. Specifying both parameters concurrently is not permitted by the server.

This system procedure returns all of the data collected by the server, unless you provide a parameter to refine the results. By viewing these statistics, you can identify irregularities that can explain slow running statements.

> **i** Note
>
> If the list of returned statements is long, then it is possible that not all of the data has been captured due to space limitations.

Privileges

You must have the MONITOR and MANAGE PROFILING privileges on the system procedure.

Side Effects

None.

Example

The following query returns performance statistics for each logged statement that has both of the statements logged:

```
SELECT *
FROM dbo.sp_find_top_statements( ) TS
INNER JOIN SYS.GTSYSPERFCACHESTMT PS ON TS.stmt_hash = PS.stmt_hash
ORDER BY TS.stmt_hash;
```

For all data, use OUTER JOIN.

Related Information

Tip: Identify the Cause of Slow Statements
sp_top_k_statements System Procedure [page 1838]

GTSYSPERFCACHESTMT System View [page 1897]
GTSYSPERFCACHEPLAN System View [page 1896]

1.6.8.119 sp_forward_to_remote_server System Procedure

Allows an application to execute a SQL statement on a remote server and retrieve any result sets generated by that statement.

Syntax

```
sp_forward_to_remote_server(
@server_name
, @sql
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify the name of the remote server the SQL statement is executed on.

@sql

Use this LONG VARCHAR parameter to specify the SQL statement to execute on the remote server.

Remarks

The SQL statement is sent verbatim to the remote server and therefore the database server does not need to able to parse the statement.

To use this system procedure, you must define the remote server with the CREATE SERVER statement.

Unlike the FORWARD TO statement, sp_forward_to_remote_server can be used within procedures. However, this stored procedure cannot be used within the FROM clause of a SELECT statement since the schema of the remote result sets is arbitrary. You can fetch remote result sets by declaring a cursor on a stored procedure that is called in the sp_forward_to_remote_server procedure.

Note

If the SQL statement returns multiple result sets, the sp_forward_to_remote_server stored procedure returns each remote result set in turn.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

There are no local side effects to executing this stored procedure; however, since the SQL statement that is executed on the remote server is arbitrary, there may be side effects on the remote server.

Example

The following example uses the sp_forward_to_remote_server stored procedure to determine the number of non-system tables in a remote SQL Anywhere 17 database called RemoteSA.

```
CALL dbo.sp_forward_to_remote_server( 'RemoteSA',
    'SELECT COUNT(*) FROM SYS.SYSTAB WHERE CREATOR NOT IN (0,3,6)' );
```

The following example uses the sp_forward_to_remote_server stored procedure to read the columns from a Microsoft Excel spreadsheet named newSalesData.

```
call dbo.sp_forward_to_remote_server( 'RemoteExcel',
    'SELECT * FROM newSalesData' );
```

Related Information

FORWARD TO Statement [page 1170]
CREATE SERVER Statement [page 962]

1.6.8.120 sp_generate_key_pair System Procedure

Creates a new RSA encryption key pair.

Syntax

```
sp_generate_key_pair system procedure(
keysize
, public_key
, private_key
, algorithm
)
```

Parameters

keysize Use this positive INTEGER parameter to indicate the key size in bits. Keys shorter than 2048 bits are not secure.

public_key Use this LONG VARCHAR out parameter to return the public key.

private_key Use this LONG VARCHAR out parameter to return the private key.

algorithm Use this optional string parameter to specify the encryption algorithm. Only 'RSA' is supported. The default is RSA.

Remarks

When using RSA key pairs, data can only be encrypted with a public key and only decrypted with a private key. Both keys are returned in PEM format.

Creating a key pair takes longer with higher key sizes but also provides much greater security and allows longer messages to be encrypted.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example creates a 2048 bit encryption key pair:

```
DECLARE @publickey LONG VARCHAR;
DECLARE @privatekey LONG VARCHAR;
CALL dbo.sp_generate_key_pair( 2048, @publickey, @privatekey );
```

Related Information

Creating Key Pairs

SECURE_SIGN_MESSAGE Function [String] [page 545]

SECURE_VERIFY_MESSAGE Function [String] [page 546]

1.6.8.121 sp_get_last_synchronize_result System Procedure

Returns information about the last synchronization initiated by the SYNCHRONIZE statement.

Syntax

```
sp_get_last_synchronize_result(
[ @conn_id
[, @complete_only ] ]
)
```

Parameters

@conn_id

Use this optional INTEGER parameter to specify the connection ID number for a connection on which the SYNCHRONIZE statement was executed. The default is NULL. If the parameter is not specified or is NULL, then the connection ID of the current connection is used.

@complete_only

Set this optional BIT parameter to 1 to have the stored procedure return information about completed synchronizations. Set the parameter to 0 to return information about synchronizations that are currently active. The default is 1.

Result Set

Column name	Data type	Description
row_id	BIGINT	The primary key of the table used to determine the order in which rows were inserted into the table.
conn_id	UNSIGNED INTEGER	The connection id number of the connection that executed the SYNCHRONIZE statement that generated this event.
result_time	TIMESTAMP	The time the event was added to the synchronize_results table. This value is affected by simulated time zone.
result_type	CHAR(128)	The type of event.

Column name	Data type	Description
parm_id	INTEGER	Each event can have zero or more parameters associated with it. The parm_id column orders the parameters associated with each event.
parm_result	LONG VARCHAR	The message text associated with the event parameter.

Remarks

To view details of past or current synchronizations, you can use the sp_get_last_synchronize_result stored procedure as an alternative to directly querying the synchronize_results and synchronize_parameters global shared temporary table. The stored procedure only returns the results of the last synchronization for the specified connection ID number. If you do not specify any parameters, the last completed synchronization on the current connection is returned.

You can also use this stored procedure to monitor the progress of a synchronization on a connection that is different from your current connection. To monitor the progress of a synchronization on a different connection:

1. Execute a SELECT CONNECTION_PROPERTY statement to determine the connection ID of your current connection.
2. Execute a SYNCHRONIZE statement using the connection ID returned by the SELECT CONNECTION_PROPERTY statement.
3. On a different connection, execute a SELECT CONNECTION_PROPERTY statement and set the `complete_only` parameter to 0. Information about the last synchronization for the specified connection is returned, even if the synchronization is incomplete.

Following is a list of events and their associated parm_id values from the synchronize_parameters table:

Event	parm_id value	Description
DBSC_EVENTTYPE_ERROR_MSG	0	The text of the error message.
	1	The message id associated with the message.
DBSC_EVENTTYPE_WARNING_MSG	0	The text of the warning message.
	1	The message id associated with the message.
DBSC_EVENTTYPE_INFO_MSG	0	The text of the information message.
	1	The message id associated with the message.
DBSC_EVENTTYPE_PROGRESS_INDEX	0	The new progress index value.
DBSC_EVENTTYPE_PROGRESS_TEXT	0	The new progress text.
DBSC_EVENTTYPE_TITLE	0	The new window title.

Event	parm_id value	Description
DBSC_EVENTTYPE_SYNC_DONE	0	The exit code from the synchronization. 0 indicates success.
DBSC_EVENTTYPE_ML_CONNECT	0	The communications protocol being used.
	1	The network protocol options being used.
DBSC_EVENTTYPE_DOWN-LOAD_COMMITTED	0	The number of insert/update operations committed.
	1	The number of delete operations committed.
DBSC_EVENTTYPE_UPLOAD_SENT	0	The number of insert operations uploaded.
	1	The number of update operations uploaded.
	2	The number of delete operations uploaded.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SELECT object-level privilege on the synchronize_results and synchronize_parameters shared global temporary tables.

Side Effects

None

Example

The following example returns information about the last synchronization that completed on the current connection.

```
CALL dbo.sp_get_last_synchronize_result();
```

The following example uses named parameters when calling the sp_get_last_synchronize_result system procedure, and returns information about the last completed synchronization that was initiated from connection ID 25.

```
CALL dbo.sp_get_last_synchronize_result(
     @conn_id=25,
     @complete_only=1);
```

Related Information

SYNCHRONIZE Statement [MobiLink] [page 1433]

1.6.8.122 sp_has_role System Procedure

Returns whether the invoker of the procedure has been granted the specified system privilege or user-defined role.

Syntax

```
sp_has_role(
rolename
[, grant_type
[, throw_error ] ]
)
```

Parameters

rolename

grant_type

Use this optional CHAR(20) parameter to specify the grant type to check for. Possible values are: ADMIN, or NO ADMIN (the default).Use this CHAR(128) parameter to specify the name of a system privilege or user-defined role to check for.

If set to ADMIN, the sp_has_role procedure checks whether the invoking user has administrative rights for the privilege or role. If set to NO ADMIN, sp_has_role checks whether the invoking user has the rights to exercise the privilege or role.

throw_error

Use this optional INTEGER parameter to specify whether to return a value indicating the outcome of the privilege check. If 1 is specified, a message is returned if the invoker has not been granted the specified privilege or role. If 0 is specified (the default), no message is returned, regardless.

Returns

Value returned	Description
1	Use this CHAR(128) parameter to specify the name of a system privilege orThe system privilege or user-defined role is granted to invoking user, and the `grant_type` check passes as well.

Value returned	Description
0 or Permission denied: you do not have permission to execute this command/procedure.	The system privilege or user-defined role is not granted to invoking user, and/or the `grant_type` check failed as well. The error message replaces the value 0 when `throw_error` is set to 1.
-1	The system privilege or user-defined role specified does not exist. No error message appears, even if `throw_error` is set to 1.

Remarks

The `throw_error` argument is useful for returning "permission denied" error messages when a user fails the permission check in a stored procedure.

Privileges

You must have EXECUTE privilege on the system procedure.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

- The following statement checks whether you have been granted the CREATE ANY PROCEDURE system privilege with no administrative rights.

  ```
  SELECT dbo.sp_has_role( 'CREATE ANY PROCEDURE' );
  ```

- The following statement checks whether you have been granted the CREATE ANY PROCEDURE system privilege with administrative rights and returns an error only if you have not been granted it.

  ```
  SELECT dbo.sp_has_role( 'CREATE ANY PROCEDURE', 'ADMIN', 1 );
  ```

- The following statement checks whether you have been granted Role_A with no administrative rights.

  ```
  SELECT dbo.sp_has_role( 'Role_A' );
  ```

- The following statement checks whether you have been granted Role_A with administrative rights and returns an error only if you have not been granted it.

```
SELECT dbo.sp_has_role( 'Role_A', 'ADMIN', 1 );
```

Related Information

Roles
sp_sys_priv_role_info System Procedure [page 1836]
sp_proc_priv System Procedure [page 1805]
sp_displayroles System Procedure [page 1768]
sp_objectpermission System Procedure [page 1796]

1.6.8.123 sp_http_listeners System Procedure

Lists the HTTP and HTTPS connection listeners used for the specified database.

Syntax

```
sp_http_listeners( database-ID )
```

Parameters

database-ID The ID of the database that the HTTP and HTTPS connection listeners are servicing. The default is the current database ID.

Result Set

Column name	Data type	Description
ip_address	VARCHAR (128)	Returns the IP address of the connection listener.
port	INTEGER	Returns the port number of the connection listener.
dbname	VARCHAR (255)	Returns NULL if the connection listener can service any database; otherwise, returns the database name.

Column name	Data type	Description
uri_prefix	LONG VARCHAR	Returns the prefix of any URI that can be serviced by the connection listener. Includes the http:// or https:// identifier, the IP address, the port number (optional), and the database name if required.

Remarks

One row appears in the result set for each HTTP and HTTPS connection listener running. A row only appears if a connection listener is available to execute web services on the specified database.

Privileges

You must have EXECUTE privilege on the system procedure.

To execute this system procedure for other databases, you must have any one of the following system privileges:

- SERVER OPERATOR
- MONITOR
- MANAGE LISTENERS

Example

Start a database server using the following command:

```
dbeng17 database1.db database2.db -xs http(port=80),http(port=8080;dbn=database1)
```

Connect to database1 and execute the following statement:

```
SELECT * FROM dbo.sp_http_listeners();
```

The database server returns a result set similar to the following:

ip_address	port	dbname	uri_prefix
127.0.0.1	80	NULL	http://127.0.0.1/database1/
127.0.0.1	8080	database1	http://127.0.0.1:8080/

If you connect to database2 and run the same statement, then the database server returns the following result set:

ip_address	port	dbname	uri_prefix
127.0.0.1	80	NULL	http://127.0.0.1/database2/

1.6.8.124 sp_list_directory System Procedure

Returns information about the files and subdirectories in a specified directory.

Syntax

```
sp_list_directory(
root_path
[, max_depth ]
)
```

Parameters

root_path

Use this LONG NVARCHAR parameter to specify the absolute or relative directory path. The relative path is relative to the working directory of the server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

max_depth

Use this optional INTEGER parameter to specify the maximum number of directories to traverse. A max_depth of NULL, 0, or a negative value results in all subdirectories of `root_path` being traversed. The default is NULL.

Result Set

Column name	Column description
`file_path`	LONG NVARCHAR - The path to a file or subdirectory within the specified directory. If `directory_path` is specified as a relative path, then the returned `file_path` is a relative value. Otherwise, `file_path` is an absolute value. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.
`file_type`	NVARCHAR(1) - F if the `file_path` value is a file or D when the `file_path` value is a directory.

Column name	Column description
file_size	UNSIGNED BIGINT - Specifies the size in bytes of the file or NULL when file_path value is a directory
owner	NVARCHAR(128) - The owner of the file or directory.
create_date_time	TIMESTAMP WITH TIME ZONE - The date and time the file or directory was created, returned in the database server's time zone.
modified_date_time	TIMESTAMP WITH TIME ZONE - The date and time the file or directory was last modified, returned in the database server's time zone.
access_date_time	TIMESTAMP WITH TIME ZONE - The date and time the file or directory was last accessed, returned in the database server's time zone.
permissions	VARCHAR(10) - The set of access permissions for the file or directory.

Remarks

None

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE system privilege.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement lists the files and subdirectories of the `c:\sqlany\samples\SQLAnywhere` directory.

```
CALL dbo.sp_list_directory('c:\\sqlany\\samples\\SQLAnywhere');
```

The following statement returns the total number of files and folders in the `c:\sqlany\samples\SQLAnywhere` and `c:\sqlany\samples\UltraLite` directories.

```
SELECT COUNT(*) dbo.FROM sp_list_directory( 'c:\\sqlany\\samples\\SQLAnywhere' )
UNION ALL
SELECT COUNT(*) FROM dbo.sp_list_directory( 'c:\\sqlany\\samples\\UltraLite' );
```

If both the `c:\sqlany\samples\SQLAnywhere` and `c:\sqlany\samples\UltraLite` directories contain the same file names, file types, and file sizes, then the following statement combines the results from two directories and gives you the total number of unique results.

```
SELECT COUNT(*) FROM
(
    SELECT REPLACE( file_path, 'c:\\sqlany\\samples\\SQLAnywhere', '' ) AS
file_path, file_type, file_size
    FROM dbo.sp_list_directory( 'c:\\sqlany\\samples\\SQLAnywhere' )
    UNION
    SELECT REPLACE( file_path, 'c:\\sqlany\\samples\\UltraLite', '' ) AS
file_path, file_type, file_size
    FROM dbo.sp_list_directory( 'c:\\sqlany\\samples\\UltraLite' )
) d;
```

Related Information

Creating Secured Feature Keys
sp_copy_directory System Procedure [page 1755]
sp_copy_file System Procedure [page 1757]
sp_create_directory System Procedure [page 1758]
sp_delete_directory System Procedure [page 1764]
sp_delete_file System Procedure [page 1765]
sp_move_directory System Procedure [page 1793]
sp_move_file System Procedure [page 1795]
Directory and File System Procedures [page 1516]

1.6.8.125 sp_list_mutexes_semaphores System Procedure

Returns information about all temporary and permanent mutexes and semaphores, including which connection is holding each mutex and whether a semaphore is being waited for.

> **i Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Syntax

```
sp_list_mutexes_semaphores( [oid] )
```

Parameters

oid (For internal use only) The unsigned bigint object ID parameter. Use the default parameter value NULL.

Result Set

Column name	Type	Column description
mutex_semaphore_id	UNSIGNED INTEGER	The ID for the object. If the object is permanent, the ID value is the same as found SYSMUTEXSEMAPHORE system view. If the object is temporary, the value is the internally-generated temporary ID.
creator	VARCHAR(128)	The owner of the mutex or semaphore
"name"	VARCHAR(128)	The name of the mutex or semaphore
"type"	VARCHAR(9)	The type of object. The value 'mutex:T' is for transaction-scope mutexes, the value 'mutex:C' is for connection-scope mutexes, and the value 'semaphore' is for semaphores.
is_temp	CHAR(1)	'Y' or 'N' indicating whether the mutex or semaphore is temporary.
is_dropped	CHAR(1)	'Y' or 'N' indicating whether the mutex or semaphore is dropped but not yet released. Y indicates that the mutex was dropped but still needs to be released, N indicates the mutex was dropped and released. This value is always N for semaphores
start_with	UNSIGNED INTEGER	The initial value of the semaphore. This value is always NULL for mutexes.
current_count	UNSIGNED INTEGER	The current value of the semaphore. This value is always NULL for mutexes.

Column name	Type	Column description
currently_owned_by	LONG VARCHAR	A comma-separated list of connection IDs for connections currently holding the mutex locked.
currently_waited_for	LONG VARCHAR	A comma-separated list of connection IDs for connections that are currently waiting for a semaphore or that are currently blocked on a mutex.

Remarks

None

Privileges

You must have EXECUTE privilege on the system procedure, and the MONITOR and UPDATE ANY MUTEX SEMAPHORE system privileges.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns information about all of the mutexes and semaphores in the database:

```
CALL dbo.sp_list_mutexes_semaphores();
```

1.6.8.126 sp_list_secure_feature_keys System Procedure

Returns a list of defined secured feature keys.

Syntax

```
sp_list_secure_feature_keys( )
```

Result Set

Column name	Data type	Description
name	VARCHAR(128)	The name of the secured feature key.
features	LONG VARCHAR	The secured features enabled by the secured feature key.

Remarks

You must have the MANAGE_KEYS feature enabled on the connection to run this procedure.

This procedures returns the names of existing secured feature keys, as well as the set of features that can be enabled by each key.

If the user has the MANAGE_FEATURES and MANAGE_KEYS features enabled, then the procedure returns a list of all secured feature keys.

If the user only has the MANAGE_KEYS feature enabled, then the procedure returns keys that have the same features or a subset of the same features that the current user has enabled.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Side Effects

None

Example

In order for the following example to work, the server must be started with the option: `-sk securefkey`.

This example enables the SYSTEM secured feature key which includes the MANAGE_FEATURES and MANAGE_KEYS features, and then uses sp_list_secure_feature_keys to obtain a list of all the current secured feature keys:

```
CALL dbo.sp_use_secure_feature_key( 'system', 'securefkey' );
CALL dbo.sp_list_secure_feature_keys( );
```

Related Information

Creating Secured Feature Keys
sp_alter_secure_feature_key System Procedure [page 1752]
sp_create_secure_feature_key System Procedure [page 1760]
sp_drop_secure_feature_key System Procedure [page 1771]
sp_use_secure_feature_key System Procedure [page 1854]

1.6.8.127 sp_login_environment System Procedure

Sets connection options when users log in.

Syntax

sp_login_environment( )

Remarks

The dbo.sp_login_environment procedure is called by the login_procedure database option by default.

Do not edit this procedure. Instead, to change the login environment, set the login_procedure option to point to a different procedure.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Related Information

login_procedure Option

1.6.8.128 sp_move_directory System Procedure

This function moves the directory pointed to by source_path to the destination pointed to by destination_path.

Syntax

```
sp_move_directory(
source_path
, destination_path
)
```

Parameters

source_path

Use this LONG NVARCHAR parameter to specify the path of the directory to be moved. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

destination_path

Use this LONG NVARCHAR parameter to specify the path where the directory is to be moved to. The directory is created if it does not already exist. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function moves all the files in the source directory to the specified directory, and then deletes the source directory.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE and WRITE FILE system privileges.

Example

The following statement moves the `SQLAnywhere` directory including its files and subdirectories to `c:\temp\SQLAnywhere`.

```
SELECT dbo.sp_move_directory('c:\\sqlany\\samples\\SQLAnywhere', 'c:\\temp\
\SQLAnywhere');
```

The original directory is removed.

The following statement restores the `SQLAnywhere` directory to its original location.

```
SELECT dbo.sp_move_directory('c:\\temp\\SQLAnywhere', 'c:\\sqlany\\samples\
\SQLAnywhere');
```

Related Information

Creating Secured Feature Keys
sp_copy_directory System Procedure [page 1755]
sp_copy_file System Procedure [page 1757]
sp_create_directory System Procedure [page 1758]
sp_delete_directory System Procedure [page 1764]
sp_delete_file System Procedure [page 1765]
sp_list_directory System Procedure [page 1786]
sp_move_file System Procedure [page 1795]
Directory and File System Procedures [page 1516]

1.6.8.129 sp_move_file System Procedure

Moves the specified file to a new directory on the computer.

Syntax

```
sp_move_file(
source_path
, destination_path
)
```

Parameters

source_path

Use this LONG NVARCHAR parameter to specify the file path, including file name, of the file to be moved. The path can be absolute or relative. A relative path is resolved to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

destination_path

Use this LONG NVARCHAR parameter to specify the destination file path, including file name, on the computer. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function moves a file from one directory to another.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE and WRITE FILE system privileges.

Example

The following statement moves the file `license.txt` to the `c:\temp` directory and gives it a new name.

```
SELECT dbo.sp_move_file('c:\\sqlany\\license.txt', 'c:\\temp\\license.bkp');
```

The file no longer exists in its original location.

The following statement restores the file to its original name and location.

```
SELECT dbo.sp_move_file('c:\\temp\\license.bkp', 'c:\\sqlany\\license.txt');
```

Related Information

Creating Secured Feature Keys
sp_copy_directory System Procedure [page 1755]
sp_copy_file System Procedure [page 1757]
sp_create_directory System Procedure [page 1758]
sp_delete_directory System Procedure [page 1764]
sp_delete_file System Procedure [page 1765]
sp_list_directory System Procedure [page 1786]
sp_move_directory System Procedure [page 1793]
Directory and File System Procedures [page 1516]

1.6.8.130 sp_objectpermission System Procedure

Generates a report on object privileges granted to a specified role or user ID, or a report on object privileges granted on a specified object or dbspace.

Syntax

```
sp_objectpermission(
object_name
, object_owner
, object_type
)
```

Parameters

object_name

The CHAR(128) name of an object or dbspace or a user or a role. If this argument is not specified, the object privileges of the current user are displayed. Default value is NULL.

object_owner

The CHAR(128) name of the object owner for the specified object name. The object privileges of the specified object owned by the specified object owner are displayed. Default value is NULL.

object_type

The CHAR(20) type of database object. If no value is specified, privileges on all object types are returned. The default value is NULL. Valid values are:

- dbspace
- function
- materialized view
- procedure
- sequence
- table (Column-level object privileges are also displayed)
- user
- view

Result Set

Column name	Data type	Description
grantor	CHAR(128)	Returns the user ID of the grantor.
grantee	CHAR(128)	Returns the user ID of the grantee.
object_name	CHAR(128)	Returns the name of the object.
owner	CHAR(128)	Returns the name of the object owner.
object_type	CHAR(20)	Returns the type of object.
column_name	CHAR(128)	Returns the name of the column.
permission	CHAR(20)	Returns the name of the privilege.
grantable	CHAR(1)	Returns a value indicating whether the privilege is grantable.

Remarks

All arguments are optional and can generate the following reports:

- If the input is an object (table, view, procedure, function, sequence, and so on), then the procedure displays a list of all users and roles that have different object privilege on the object.
- If the input is a role or user, then the procedure displays a list of all object privileges granted to the role or user.
- If the input is a dbspace name, then the procedure displays a list of all users and roles which has CREATE privilege on the specified dbspace.

When executing sp_objectpermission to display object privileges for a user or a role, the object privileges that are inherited through role grants are also displayed. By default the object_type is NULL and the object privileges for all existing object types matching the specified object name are displayed.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Privileges

You must have EXECUTE privilege on the system procedure.

To call this procedure on another user ID, or on an object owned by another user ID, you must also have the MANAGE ANY OBJECT PRIVILEGE system privilege.

To execute this procedure for a dbspace, you must also have the MANAGE ANY DBSPACE system privilege.

Side Effects

None

Example

The following statement returns the object-level privileges granted to the DIAGNOSTICS system role. For the purposes of this example, the results have been truncated.

```
CALL dbo.sp_objectpermission( 'SA_DEBUG' );
```

grantor	grantee	object_name	owner	object_type	column_name	permission	grantable
(NULL)	PUBLIC	sa_flush_statistics	dbo	PROCEDURE	(NULL)	EXECUTE	N
(NULL)	PUBLIC	sa_audit_string	dbo	PROCEDURE	(NULL)	EXECUTE	N
(NULL)	PUBLIC	sa_enable_auditing_type	dbo	PROCEDURE	(NULL)	EXECUTE	N

grantor	grantee	object_name	owner	object_type	col-umn_name	permission	grantable
(NULL)	PUBLIC	sa_disa-ble_audit-ing_type	dbo	PROCEDURE	(NULL)	EXECUTE	N
...	...	...	...	...	...	...	...

The results show that there are many procedures on which the SA_DEBUG system role has object-level permissions.

The following statement returns the object-level privileges granted to the ml_server user.

```
CALL dbo.sp_objectpermission( 'ml_server' );
```

The following statement returns the object-level privileges on the system dbspace.

```
CALL dbo.sp_objectpermission( object_name='system', object_type='dbspace' );
```

Related Information

Roles
Viewing the Roles and Privileges for a User or Role (SQL)
Viewing the Roles and Privileges for a User or Role (SQL Central)
sp_sys_priv_role_info System Procedure [page 1836]
sp_has_role System Procedure [page 1782]
sp_proc_priv System Procedure [page 1805]
sp_displayroles System Procedure [page 1768]

1.6.8.131 sp_parse_json System Procedure

Returns a representation of JSON data using SQL data types.

Syntax

```
sp_parse_json(
  var
 , jsonstring
  [, maxdepth ]
)
```

Parameters

var

Use this LONG VARCHAR parameter to specify the name of the connection-level variable to create. The type of variable is determined at execution time. The variable is created if it does not exist.

jsonstring

Use this LONG NVARCHAR parameter to specify a string representation of the JSON data structure.

maxdepth

Use this optional INTEGER parameter to specify the maximum nesting level of sets. Nodes below the stipulated depth are not processed. Instead, they are returned as a fragment of JSON. The default value is 100.

Remarks

This procedure processes a JSON object and returns the processed data as a SQL ROW or ARRAY variable. The type of return variable is determined when the procedure is executed. In most cases, the return variables are either ROW or ARRAY SQL data types.

The sp_parse_json system procedure returns a VARCHAR fragment for its base ARRAY or OBJECT when subsequent instances return a different number of nodes.

JSON object identifiers must comply with the identifier rules defined in the database server. As well, the database server enforces the same limits for JSON data types as it does for the underlying ROW and ARRAY data types.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example parses a JSON string into a SQL ROW variable and creates a result set from it.

```
CREATE OR REPLACE VARIABLE sql_row_var ROW( a INTEGER, b INTEGER );
SET sql_row_var.a = -1;
SET sql_row_var.b = -1;
CALL sp_parse_json ( 'sql_row_var', '{a:1, b:2}' );
SELECT sql_row_var.a, sql_row_var.b;
```

The following example sets up a table with some data to generate a JSON string (`[{"name":"Frank","age":51},{"name":"Bill","age":22},{"name":"Jackie","age":37}]`) that can be parsed. Then it parses the JSON string into a SQL array variable and creates a result set from it.

```
BEGIN
    DECLARE json_data LONG VARCHAR;
    DECLARE LOCAL TEMPORARY TABLE test (
        name AS VARCHAR(64),
        age AS INT);
    INSERT INTO test (name, age) VALUES ('Frank',51);
    INSERT INTO test (name, age) VALUES ('Bill',22);
    INSERT INTO test (name, age) VALUES ('Jackie',37);
    SELECT * INTO json_data FROM test FOR JSON RAW;
    DROP VARIABLE IF EXISTS sql_array;
    CALL sp_parse_json ( 'sql_array', json_data, 2 );
    SELECT sql_array [[row_num]].name AS name, sql_array [[row_num]].age AS age
        FROM sa_rowgenerator ( 1, CARDINALITY(sql_array) );
END;
```

The JSON string in the previous example contains a set of sets. Since `maxdepth` is specified, it must be 2 or greater to properly parse the string.

The following example parses a JSON string with nested sets into a SQL array variable and creates a result set from it.

```
CREATE OR REPLACE VARIABLE arrayvar ARRAY OF ROW(
        a VARCHAR(32),
        b ARRAY OF ROW( b1 LONG NVARCHAR, b2 LONG NVARCHAR),
        c BIT,
        d NUMERIC(5,2)
);
CALL sp_parse_json( 'arrayvar',
'[{a:"first", b:[{b1:"Hello", b2:"John"},{b1:"Goodbye", b2:"Mary"}], c:true, d:
12.34},' ||
' {a:"second", b:[{b1:"Bonjour", b2:"Jean"},{b1:"Au revoir", b2:"Marie"}],
c:false, d:56.78}]' );
SELECT arrayvar[[x.row_num]].a AS a,
    arrayvar[[x.row_num]].b[[y.row_num]].b1 AS b1,
    arrayvar[[x.row_num]].b[[y.row_num]].b2 AS b2,
    arrayvar[[x.row_num]].c AS c,
    arrayvar[[x.row_num]].d AS d
    FROM sa_rowgenerator(1,CARDINALITY(arrayvar)) AS x,
         sa_rowgenerator(1,CARDINALITY(arrayvar[[1]].b)) AS y;
```

Related Information

ROW and ARRAY Composite Data Types [page 185]
Identifiers [page 6]
ROW Constructor [Composite] [page 534]
ARRAY Constructor [Composite] [page 243]

1.6.8.132 sp_plancache_contents System Procedure

Returns the contents of a plan cache for a connection, including statistics indicating the average, minimum, and maximum execution times, and the number of times cursors have been built for a statement.

Syntax

```
sp_plancache_contents( [ connidparm ] )
```

Parameters

connidparm

This optional INTEGER parameter is the connection ID of a connection to the current database. The default is NULL, in which case information about the current connection's plan cache is returned..

Result Set

Column name	Data type	Description
stmt_type	CHAR(1)	A value that indicates the type of statement. Possible values are: **B** Bypass statement **C** Client statement **E** Event **M** Temporary procedure or batch **P** Procedure **T** Trigger

Column name	Data type	Description
definition_id	UNSIGNED INTEGER	For bypass statements, the associated table_id found in the SYSTAB system view. For client statements, an ID that uniquely identifies the statement. This ID matches the value printed with the corresponding statement text in the request level log, if logging of statement text is enabled. For event statements, the associated event_id value found in the SYSEVENT system view. For temporary procedures, an ID that uniquely identifies the procedure on the connection. For batches, this field is NULL. For procedure statements, the associated proc_id value found in the SYSPROCEDURE system view. For trigger statements, the associated trigger_id value found in the SYSTRIGGER system view.
definition_position	UNSIGNED INTEGER	A positional offset of the statement within the definition. For bypass and client statements this field is NULL.
definition_scope	SMALLINT	The scope of the statement context. For bypass and client statements this field is NULL.
bypass_type	CHAR(1)	For bypass statements, a value that indicates the bypass statement type. Possible values are: **D** DELETE **I** INSERT **S** SELECT **U** UPDATE

Column name	Data type	Description
status	CHAR(1)	A value that indicates the current state of the cache entry. Possible values are: **D** Caching is disabled for this statement. **P** A plan is cached for this statement. **R** Ready to cache a plan on the next execution of this statement. **T** Training to determine whether to cache a plan for this statement.
plan_type	CHAR(1)	For internal use only.
plan_signature	UNSIGNED BIGINT	For internal use only.
PSID	UNSIGNED BIGINT	For internal use only.
build_count	UNSIGNED INTEGER	The number of times a cursor has been built for this statement.
build_avg_msec	DOUBLE	The average time required to build a cursor for this statement, in milliseconds.
reusable_count	UNSIGNED INTEGER	The number of executions of a reusable cursor for this statement.
reusable_avg_msec	DOUBLE	The average elapsed time for an execution of a reusable cursor for this statement, in milliseconds.
reusable_min_msec	DOUBLE	The minimum elapsed time for an execution of a reusable cursor for this statement, in milliseconds.
reusable_max_msec	DOUBLE	The maximum elapsed time for an execution of a reusable cursor for this statement, in milliseconds.
nonreusable_count	UNSIGNED INTEGER	The number of executions of a nonreusable cursor for this statement.
nonreusable_avg_msec	DOUBLE	The average elapsed time for an execution of a nonreusable cursor for this statement, in milliseconds.
nonreusable_min_msec	DOUBLE	The minimum elapsed time for an execution of a nonreusable cursor for this statement, in milliseconds.
nonreusable_max_msec	DOUBLE	The maximum elapsed time for an execution of a nonreusable cursor for this statement, in milliseconds.

Remarks

Use this system procedure to examine the current entries in a connection's plan cache. This can be helpful for choosing an appropriate value for the max_plans_cached option, which governs plan cache size. The statistics about build and execution times can also be used to identify statements for which plan caching is causing performance issues. In this case, plan caching can either be disabled for all statements by setting the max_plans_cached option to 0 (zero), or by selectively disabling caching for a statement by adding the FORCE OPTIMIZATION clause to the SQL text.

Privileges

You must have EXECUTE privilege on the system procedure. To return the plan cache contents for a connection other than the current connection, you must also have the MANAGE CACHED PLANS system privilege.

Side Effects

None

Example

The following statement returns the plan cache entries for the current connection:

```
CALL dbo.sp_plancache_contents();
```

Related Information

Plan Caching
Eligibility to Skip Query Processing Phases
max_plans_cached Option

1.6.8.133 sp_proc_priv System Procedure

Returns the list of system privileges required to run a system procedure.

Syntax

```
sp_proc_priv( [proc_name] )
```

Parameters

proc_name

This CHAR(128) parameter specifies the name of the system procedure to return privileges for. If `proc_name` is not specified, the privileges required for all system procedures are returned. The default is NULL.

Result Set

Column name	Data type	Description
`proc_name`	CHAR(128)	The name of the system procedure.
`privilege`	LONG VARCHAR	The list of privileges required to run the system procedure.

Remarks

In the result set, if a number of privileges separated by a comma is listed for a system procedure, then it implies that any one of those privileges would suffice. If multiple rows are displayed for a system procedure, then one privilege from each row is required to execute the system procedure.

If sp_proc_priv is invoked without `proc_name`, it returns privilege information for all system procedures that require privileges. System procedures that do not require privileges are not included in the results.

sp_proc_priv returns only the privileges that guarantee that a system procedure will pass a permissions check. However, in some contexts, there may be additional privileges that would allow a system procedure to pass a permissions check. Those privileges are not returned by sp_proc_priv.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns the privileges required to run the sa_table_fragmentation system procedure. Since only one row is returned, any one of the privileges in the comma separated list is sufficient to run sa_table_fragmentation.

```
CALL dbo.sp_proc_priv( 'sa_table_fragmentation' );
```

proc_name	privilege
sa_table_fragmentation	MANAGE ANY STATISTICS, MONITOR

The following example returns the privileges for the sa_install_feature system procedure. Since multiple rows are returned, one privilege from each row is required to run sa_install_feature.

```
CALL dbo.sp_proc_priv( 'sa_install_feature' );
```

proc_name	privilege
sa_install_feature	SELECT ANY TABLE
sa_install_feature	MANAGE ANY SPATIAL OBJECT
sa_install_feature	MANAGE ANY OBJECT PRIVILEGE
sa_install_feature	CREATE ANY PROCEDURE, CREATE ANY OBJECT

Related Information

Roles
sp_sys_priv_role_info System Procedure [page 1836]
sp_has_role System Procedure [page 1782]
sp_displayroles System Procedure [page 1768]
sp_objectpermission System Procedure [page 1796]

1.6.8.134 sp_property_history System Procedure

Returns values for all database server properties tracked by the database.

Syntax

```
sp_property_history( property, min_ticks )
```

Parameters

property Use this VARCHAR(255) to specify the name of the database server property to report. If NULL, then all currently monitored properties are reported. The default is NULL.

min_ticks Specify a tick value to return all recorded property values with a ticks value that is equal to or greater than the specified tick value. The default is NULL.

Result Set

Column name	Data type	Description
name	VARCHAR(255)	The name of the database server property.
ticks	UNSIGNED BIGINT	A monotonically increasing value that chronologically orders property values.
time_recorded	TIMESTAMP WITH TIME ZONE	The system time when this value was recorded.
time_delta	UNSIGNED INTEGER	The number of milliseconds since the previous recording, independent of system time.
value	DOUBLE	The current value of the database server property.
value_delta	DOUBLE	The change in the database property value since the previous recording.

Remarks

This system procedure returns results for database server properties being tracked by any database running on the database server, as well as by the -phl database server option. The database server uses ticks, measured by your computer's system clock, to track the chronological order in which property values are recorded. Each recorded value has a tick value that increases monotonically, along with an associated timestamp measured in GMT.

If `property-name` is NULL, then all database server property values are returned.

If `min_ticks` is NULL, then all property values for the selected properties (or all properties if `property-name` is NULL) are returned.

If the database is restarted, then property history data is only kept for properties currently being tracked by another running database.

If the database server is restarted, then property history data and tracking settings are lost. Desired tracking settings must be re-supplied.

Database-specific property tracking settings are also lost if all [of] the following are true:

- The database is restarted.
- No other database running on the database server is tracking the database server property.
- The database server property is not being tracked at the database server level.

> → Tip
>
> To maintain database-specific tracking settings, create a database start-up event to mimic the persistence of these settings.

Privileges

You must have EXECUTE privilege on the system procedure.

You must have the MANAGE ANY PROPERTY HISTORY system privilege.

Side Effects

None

Example

To list all of the recorded database server property values in descending order, execute the following statement:

```
SELECT * FROM dbo.sp_property_history( )
    ORDER BY ticks desc;
```

To pivot the property history information so that each row shows the delta changes per tick/time, execute the following query:

```
BEGIN
    DECLARE @props VARCHAR(120) ARRAY;
    SELECT *
        INTO #propdata
        FROM dbo.sp_property_history();
    SET @props = ( SELECT ARRAY_AGG( DISTINCT name ORDER BY name ) FROM
#propdata );
    SELECT *
        FROM ( SELECT name, ticks, time_recorded, time_delta, value_delta FROM
#propdata ) mysourcedata
        PIVOT ( SUM( value_delta ) delta FOR NAME IN @props ) mydata
        ORDER BY ticks DESC;
END;
```

To pivot the property history information so that each row shows the value changes per tick/time, execute the following query:

```
BEGIN
    DECLARE @props VARCHAR(120) ARRAY;
    SELECT *
        INTO #propdata
        FROM dbo.sp_property_history();
    SET @props = ( SELECT ARRAY_AGG( DISTINCT name ORDER BY name ) FROM
#propdata );
    SELECT *
        FROM ( SELECT name, ticks, time_recorded, value FROM #propdata )
mysourcedata
        PIVOT ( SUM( value ) value FOR NAME IN @props ) mydata
        ORDER BY ticks DESC;
END;
```

Related Information

sa_db_option System Procedure [page 1566]
sa_server_option System Procedure [page 1698]
PROPERTY_IS_TRACKABLE Function [System] [page 494]
-sf Database Server Option
sa_server_option System Procedure [page 1698]
sa_db_option System Procedure [page 1566]
-phl Database Server Option
-phs Database Server Option
List of Database Server Properties
List of Database Properties
System Privileges

1.6.8.135 sp_read_db_pages System Procedure

Reads the specified pages into the cache.

Syntax

```
sp_read_db_pages( dbspace_id [, page_bitmap [, in_background [,
limit_to_cache_size ] ] ] )
```

Parameters

dbspace_id

Use this SMALLINT parameter to specify the dbspace to read pages from.

page_bitmap Use this LONG VARBIT parameter to specify which pages to read into the cache. Specifying 1 indicates that the page should be read. If this parameter is NULL (default), then all pages from the specified dbspace are read into the cache. This value is a string that is returned from the sp_db_cache_contents system procedure.

in_background

Use this BIT parameter to specify whether the pages should be loaded in the background with minimal impact on other running operations.

limit_to_cache_size

Use this BIT parameter to indicate whether to limit the number of pages read to the number of pages that can be stored in the cache. Specify 1 (default) to raise an error if more pages would be read than could fit into the cache.

Remarks

This procedure reads pages from a specified dbspace into the cache.

When executing this system procedure, only pages that are not already present in the cache are read in. If the cache cannot fit all of the requested pages, then an error is returned.

Privileges

You must have EXECUTE privilege on the system procedure. You must have the SERVER OPERATOR system privilege.

Side Effects

None

Example

1. Create a dbspace by executing the following statement:

```
CREATE DBSPACE mydbs
AS 'C:\\mydb\\mydbs.db';
```

2. Create a table in the new dbspace and add some data by executing the following statements:

```
CREATE TABLE mytable( col1 INT, col2 CHAR(128) ) IN mydbs;
```

```
INSERT INTO mytable
    SELECT column_id, column_name
```

```
    FROM sys.syscolumn;

COMMIT;
```

3. Execute the following statement to determine the ID of the new dbspace:

```
SELECT * FROM SYS.SYSDBSPACE WHERE dbspace_name = 'mydbs';
```

   Note the ID of the new dbspace and that the saved_cache_pages column is NULL.
4. Verify which pages from the new dbspace are currently in the cache by executing the following statement:

```
SELECT * FROM dbo.sp_db_cache_contents( );
```

   The statement above returns information for all dbspaces in the database. To restrict the results to the new mydbs dbspace, execute the following statement, where `mydbs-ID` is the ID of the new mydbs dbspace that was obtained from SYS.SYSDBSPACE:

```
SELECT * FROM dbo.sp_db_cache_contents( mydbs-ID );
```

5. Save the current steady state of the database by executing the following statement:

```
ALTER DATABASE SAVE CACHE;
```

6. Later, after numerous other transactions have been executed and the cache has undergone significant change, you want to return to the saved steady state. You can either read all the steady state pages for all dbspaces into the cache or read all the steady state pages for a specific dbspace into the cache.
   To read all the steady state pages for all dbspaces into the cache, execute the following statement:

```
ALTER DATABASE RESTORE CACHE;
```

   If you would rather read all the steady state pages for one dbspace (for example, the new dbspace mydbs), then execute a series of statements similar to the following:

```
CREATE VARIABLE cache_pages LONG VARBIT;
```

```
SELECT saved_cache_pages
    INTO cache_pages
    FROM SYS.SYSDBSPACE
    WHERE dbspace_name='mydbs';
```

```
CALL dbo.sp_read_db_pages( mydbs-ID, cache_pages );
```

Related Information

Cache Warming
sp_db_cache_contents System Procedure [page 1762]

1.6.8.136 sp_read_etd System Procedure

Reads the specified event trace data (ETD) file and returns the contents of the file as a set of rows.

Syntax

```
sp_read_etd(
filename
[,event_names=event-name[,...]]
[,host_names=host-name[,...]]
[,severity_level=integer]
[,record_start=integer]
[,record_end=integer]
[,timestamp_start=timestamp-with-timezone]
[,timestamp_end=timestamp-with-timezone]
[,regex ]=regular-expression[,...]]
)
```

Parameters

filename

This ARRAY of LONG NVARCHAR values specifies the full paths and names of the ETD files to be processed. When multiple files are specified, the function aggregates the results from all specified ETD files and returns the entries in ascending order by timestamp. Relative paths are interpreted relative to the working directory of the database server. If disk sandboxing is enabled, then paths are restricted to the directory where the main database file is located and any subdirectories of that directory.

event_names

This LONG NVARCHAR provides a comma-separated list of events for which data will be returned.

host_names

This LONG NVARCHAR provides a comma-separated list of hosts that have generated events.

severity_level

This LONG NVARCHAR specifies the maximum severity level to retrieve. A NULL returns all severity levels. The severity levels are:

Severity level	Value
ALWAYS	0
CRITICAL	50
ERROR	100
WARING	150
INFORMATION	200
DEBUG	250

record_start

This UNSIGNED INT identifies the first record number to retrieve. A value of NULL is treated as a 1. The first record number is 1.

record_end

Use this UNSIGNED INT to identify the last record number to retrieve. A value of NULL means return all records from the starting record to the last available record.

timestamp_start

Use this TIMESTAMP WITH TIME ZONE to identify the starting timestamp.

timestamp_end

Use this TIMESTAMP WITH TIME ZONE to identify the ending timestamp.

regex

Use this LONG NVARCHAR to specify a regular string expression to filter event fields containing string values. The filtering is treated as a logical AND operation. For parameters consisting of comma-separated lists, the filtering is treated as a logical OR operation.

Result Set

This procedure returns a result set comprised of rows, as follows:

Column	Data type	Description
timestamp	TIMESTAMP WITH TIME ZONE	The time the event occurred
pid	UNSIGNED INT	The process ID that generated the event
tid	UNSIGNED INT	The thread ID that generated the event
host_name	LONG NVARCHAR	The host that generated the event
os	LONG NVARCHAR	The operating system name
processor	LONG NVARCHAR	The processor type
severity	LONG NVARCHAR	The severity level of the event
severity_number	TINYINT	The severity level number
event_name	LONG NVARCHAR	The name of the event
event_data	LONG NVARCHAR	An XML document that describes the fields of the event
record_id	UNSIGNED BIGINT	The order that the event appears in the ETD file. The record_id starts with 0 and is monotonously increasing, with 0 being the first record in the ETD file.

Remarks

The format of the `event_data` XML document is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<event_data num_fields="X">
    <field name="Y" type="YY">Z</field>
     ...
</event_data>
```

X is the number of fields for the event, Y is the field name, YY is the field type, and Z is the data for that field. Valid values for field types are: integer (for integer sizes), string, binary, float, and double. Binary data is shown in base64 encoding.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE system privilege.

Example

To call the sp_read_etd system procedure, use the OPENXML operator to retrieve `event_data` of a single type of event as columns. The following statement returns the timestamp from the sp_read_etd system procedure and two values (ID and INFO) stored in `event_data` as a single result set:

```
SELECT T1.timestamp, T2.ID, T2.INFO
FROM dbo.sp_read_etd( Array ('C:\\test\\trace1.etd' ), event_names = 'nop' ) AS
T1,
     OPENXML ( T1.event_data, '/event_data' )
         WITH (
                ID LONG VARCHAR 'field[@name="id"]/text()',
                INFO LONG VARCHAR 'field[@name="info"]/text()'
               ) AS T2;
```

timestamp	ID	INFO
2013-09-10 14:10:33.089-04:00	1	abc
2013-09-10 14:10:36.054-04:00	2	def

Use the OPENXML operator and LIST function to retrieve `event_data` for various kinds of events in a single column using the following statement:

```
SELECT
    T1.timestamp, T1.event_name, list( id.FIELD + '=' + id.TEXT ) as data
FROM dbo.sp_read_etd( Array( 'C:\\test\\trace3.etd' ) ) AS T1,
    OPENXML ( T1.event_data, '/event_data/field' )
        WITH ( FIELD LONG VARCHAR '@name', TEXT LONG VARCHAR 'text()' ) AS id,
GROUP BY T1.timestamp, T1.event_name, T1.event_data;
```

timestamp	event_name	data
2013-09-10 12:10:13.059-04:00	EVENT2	id=1,desc=abc
2013-09-10 14:10:46.054-04:00	EVENT1	id=2,location=XYZ,other_information=abc

Use a regular expression to filter event fields containing string values. The following statement returns all event fields containing the string 'abc' in the event data column:

```
SELECT *
FROM dbo.sp_read_etd( Array( 'C:\\test\\trace1.etd' ), regex = '.*abc.*' );
```

Related Information

Event Tracing
Viewing the Contents of the Event Trace Data (ETD) Diagnostic Log File
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.6.8.137 sp_remote_columns System Procedure

Produces a list of the columns in a remote table, and a description of their data types.

Syntax

```
sp_remote_columns(
@server_name
, @table_name
[, @table_owner
[, @table_qualifier ] ]
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify a string containing the server name as specified by the CREATE SERVER statement.

@table_name

Use this CHAR(128) parameter to specify the name of the remote table.

@table_owner

Use this optional CHAR(128) parameter to specify the owner of `table_name`. The default is %.

@table_qualifier

Use this optional CHAR(128) parameter to specify the name of the database in which `table_name` is located. The default is %.

Result Set

Column name	Data type	Description
database	CHAR(128)	The database name.
owner	CHAR(128)	The database owner name.
table_name	CHAR(128)	The table name.
column_name	CHAR(128)	The name of a column.
domain_id	SMALLINT	An INTEGER that indicates the data type of the column.
width	INTEGER	The meaning of this column depends on the data type. For character types, width represents the number of characters.
scale	SMALLINT	The meaning of this column depends on the data type. For NUMERIC data types, scale is the number of digits after the decimal point.
nullable	SMALLINT	If NULL column values are allowed, the value is 1. Otherwise, the value is 0.
base_type_str	CHAR(4096)	The annotated type string representing the physical type of the column.

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

If you are entering a CREATE EXISTING TABLE statement and you are specifying a column list, it may be helpful to get a list of the columns that are available on a remote table. sp_remote_columns produces a list of the columns on a remote table and a description of their data types. If you specify a database, you must either specify an owner or provide the value NULL.

If the table does not exist on the remote server, the procedure returns an empty result set.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Standards

N/A

Example

The following example returns information about the columns in the ULProduct table on the remote SQL Anywhere database server named RemoteSA. The table owner is DBA.

```
CALL dbo.sp_remote_columns( 'RemoteSA', 'ULProduct', 'DBA', null );
```

The following example returns information about the columns in the SYSOBJECTS table in the Adaptive Server Enterprise database Production using the remote server named RemoteASE. The table owner is unspecified.

```
CALL dbo.sp_remote_columns( 'RemoteASE', 'sysobjects', null, 'Production' );
```

The following example returns information about the columns in the Customers table in the Microsoft Access database `c:\users\me\documents\MyAccesDB.accdb` using the remote server MyAccessDB. The Microsoft Access database does not have a table owner so NULL is specified.

```
CALL dbo.sp_remote_columns( 'MyAccessDB', 'Customers', null, 'c:\\users\\me\
\documents\\MyAccesDB.accdb' );
```

Related Information

Remote Data Access
Server Classes for Remote Data Access
CREATE SERVER Statement [page 962]

1.6.8.138 sp_remote_exported_keys System Procedure

Provides information about tables with foreign keys on a specified primary table.

Syntax

```
sp_remote_exported_keys(
@server_name
```

```
, @table_name
[, @table_owner
[, @table_qualifier ] ]
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify the server the primary table is located on.

@table_name

Use this CHAR(128) parameter to specify the table containing the primary key.

@table_owner

Use this optional CHAR(128) parameter to specify the primary table's owner. The default is '%'.

@table_qualifier

Use this optional CHAR(128) parameter to specify the database containing the primary table. The default is '%'.

Result Set

Column name	Data type	Description
pk_database	CHAR(128)	The database containing the primary key table.
pk_owner	CHAR(128)	The owner of the primary key table.
pk_table	CHAR(128)	The primary key table.
pk_column	CHAR(128)	The name of the primary key column.
fk_database	CHAR(128)	The database containing the foreign key table.
fk_owner	CHAR(128)	The foreign key table's owner.
fk_table	CHAR(128)	The foreign key table.
fk_column	CHAR(128)	The name of the foreign key column.
key_seq	SMALLINT	The key sequence number.
fk_name	CHAR(128)	The foreign key name.
pk_name	CHAR(128)	The primary key name.

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

This procedure provides information about the remote tables that have a foreign key on a particular primary table. The result set for the sp_remote_exported_keys system procedure includes the database, owner, table, column, and name for both the primary and the foreign key, and the foreign key sequence for the foreign key columns. The result set may vary because of the underlying ODBC and JDBC calls, but information about the table and column for a foreign key is always returned.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns information about the foreign key relationships in the ULEmployee table on the remote server named RemoteSA:

```
CALL dbo.sp_remote_exported_keys( 'RemoteSA', 'ULEmployee', 'DBA' );
```

Related Information

Foreign Keys
CREATE SERVER Statement [page 962]

1.6.8.139 sp_remote_imported_keys System Procedure

Provides information about remote tables with primary keys that correspond to a specified foreign key.

Syntax

```
sp_remote_imported_keys(
@server_name
```

```
, @table_name
[, @table_owner
[, @table_qualifier ] ]
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify the server the foreign key table is located on. A value is required for this parameter.

@table_name

Use this CHAR(128) parameter to specify the table containing the foreign key. A value is required for this parameter.

@table_owner

Use this optional CHAR(128) parameter to specify the foreign key table's owner. The default is '%'.

@table_qualifier

Use this optional CHAR(128) parameter to specify the database containing the foreign key table. The default is '%'.

Result Set

Column name	Data type	Description
pk_database	CHAR(128)	The database containing the primary key table.
pk_owner	CHAR(128)	The owner of the primary key table.
pk_table	CHAR(128)	The primary key table.
pk_column	CHAR(128)	The name of the primary key column.
fk_database	CHAR(128)	The database containing the foreign key table.
fk_owner	CHAR(128)	The foreign key table's owner.
fk_table	CHAR(128)	The foreign key table.
fk_column	CHAR(128)	The name of the foreign key column.
key_seq	SMALLINT	The key sequence number.
fk_name	CHAR(128)	The foreign key name.
pk_name	CHAR(128)	The primary key name.

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

Foreign keys reference a row in a separate table that contains the corresponding primary key. This procedure allows you to obtain a list of the remote tables with primary keys that correspond to a particular foreign table. The sp_remote_imported_keys result set includes the database, owner, table, column, and name for both the primary and the foreign key, and the foreign key sequence for the foreign key columns. The result set may vary because of the underlying ODBC and JDBC calls, but information about the table and column for a primary key is always returned.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns the tables with primary keys that correspond to a foreign key on the ULOrder table on the remote server named RemoteSA:

```
CALL dbo.sp_remote_imported_keys( 'RemoteSA', 'ULOrder', 'DBA' );
```

Related Information

Foreign Keys
CREATE SERVER Statement [page 962]

1.6.8.140 sp_remote_pcols System Procedure

Produces a list of the columns in a remote table, and a description of their data types.

Syntax

```
sp_remote_pcols(
```

```
@server_name
, @sp_name
[, @sp_owner
[, @sp_qualifier ] ]
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify a string containing the server name as specified by the CREATE SERVER statement.

@sp_name

Use this CHAR(128) parameter to specify the name of the remote table.

@sp_owner

Use this optional CHAR(128) parameter to specify the owner of `sp_name`. The default is '%'.

@sp_qualifier

Use this optional CHAR(128) parameter to specify the name of the database in which `sp_name` is located. The default is '%'.

Result Set

Column name	Data type	Description
database	CHAR(128)	The database name.
owner	CHAR(128)	The database owner name.
proc_name	CHAR(128)	The stored procedure name.
parm_name	CHAR(128)	The name of the parameter or result set column.
parm_mode	CHAR(10)	The mode of the parameter or result set column (IN, OUT, INOUT, RESULT).
domain_id	SMALLINT	An INTEGER which indicates the data type of the parameter or result set column.
width	INTEGER	The meaning of this column depends on the data type. For character types width represents the number of characters.
scale	SMALLINT	The meaning of this column depends on the data type. For NUMERIC data types scale is the number of digits after the decimal point.

Column name	Data type	Description
nullable	SMALLINT	If NULL parameter values are allowed, the value is 1. Otherwise the value is 0.

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

If you are entering a CREATE PROCEDURE ... AT statement and you are specifying a parameter list or want information about any result set that may be returned, it may be helpful to get a list of the parameters and result set columns that are available for a remote stored procedure. sp_remote_pcols produces a list of the parameters and result set columns of a remote stored procedure and a description of their data types. If you specify a database, you must either specify an owner or provide the value NULL.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Standards

N/A

Example

The following example returns information about the parameters and result set columns of the ULOrderDownload stored procedure in the remote SQL Anywhere database server named RemoteSA. The stored procedure owner is DBA.

```
CALL dbo.sp_remote_pcols('RemoteSA', 'ULOrderDownload', 'DBA');
```

The following example returns information about the parameters and result set columns of the col_name stored procedure in the Adaptive Server Enterprise database Production using the remote server named RemoteASE. The remote procedure owner is unspecified.

```
CALL dbo.sp_remote_pcols( 'RemoteASE', 'col_name', null, 'Production' );
```

Related Information

Remote Data Access
Server Classes for Remote Data Access
CREATE SERVER Statement [page 962]

1.6.8.141 sp_remote_primary_keys System Procedure

Provides primary key information about remote tables using remote data access.

Syntax

```
sp_remote_primary_keys(
  @server_name
 , @table_name
  [, @table_owner
  [, @table_qualifier ] ]
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify the remote server name.

@table_name

Use this CHAR(128) parameter to specify the name of the remote table.

@table_owner

Use this optional CHAR(128) parameter to specify the owner of the remote table. The default is '%'.

@table_qualifier

Use this optional CHAR(128) parameter to specify the name of the remote database. The default is '%'.

Result Set

Column name	Data type	Description
database	CHAR(128)	The name of the remote database.
owner	CHAR(128)	The owner of the table.
table_name	CHAR(128)	The name of the table.
column_name	CHAR(128)	The name of the primary key column.
key_seq	SMALLINT	The primary key sequence number.
pk_name	CHAR(128)	The primary key name.

Remarks

This system procedure provides primary key information about remote tables using remote data access.

Because of differences in the underlying ODBC calls, the information returned differs slightly from the catalog/database value depending upon the remote data access class that is specified for the server.

Privileges

You must have EXECUTE privilege on the system procedure.

Standards

N/A

Side Effects

None

Example

The following example returns information about the primary keys in tables owned by DBA in a SQL Anywhere remote server named RemoteSA.

```
CALL dbo.sp_remote_primary_keys( 'RemoteSA', null, 'DBA' );
```

To get a list of the primary keys in all the tables owned by Fred in the production database in an Adaptive Server Enterprise server named RemoteASE:

```
CALL dbo.sp_remote_primary_keys( 'RemoteASE', null, 'Fred', 'production' );
```

Related Information

Remote Data Access
Server Classes for Remote Data Access
CREATE SERVER Statement [page 962]

1.6.8.142 sp_remote_procedures System Procedure

Returns a list of the procedures on a remote server.

Syntax

```
sp_remote_procedures(
  @server_name
  [, @sp_name
  [, @sp_owner
  [, @sp_qualifier ] ] ]
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify the remote server name.

@sp_name

Use this optional CHAR(128) parameter to specify the remote stored procedure name. The default is '%'.

@sp_owner

Use this optional CHAR(128) parameter to specify the owner of the remote stored procedure. The default is '%'.

@sp_qualifier

Use this optional CHAR(128) parameter to specify the database in which `sp_name` is located. The default is '%'.

Result Set

Column name	Data type	Description
database	CHAR(128)	The name of the remote database.
owner	CHAR(128)	The name of the stored procedure owner.
proc_name	CHAR(128)	The name of the stored procedure.

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

It may be helpful when you are configuring your database server to get a list of the remote stored procedures available on a particular server. This procedure returns a list of the stored procedures on a server.

The procedure accepts four parameters. If a stored procedure, owner, or database name is given, the list of stored procedures will be limited to only those that match the arguments.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example returns information about the stored procedures owned by DBA in a SQL Anywhere remote server named RemoteSA.

```
CALL dbo.sp_remote_procedures( 'RemoteSA', null, 'DBA' );
```

The following example returns information about the stored procedures owned by Fred in the production database in an Adaptive Server Enterprise remote server named RemoteASE:

```
CALL dbo.sp_remote_procedures( 'RemoteASE', null, 'Fred', 'production' );
```

Related Information

Remote Data Access
Server Classes for Remote Data Access
CREATE PROCEDURE Statement [page 913]
CREATE SERVER Statement [page 962]

1.6.8.143 sp_remote_tables System Procedure

Returns a list of the tables on a server.

Syntax

```
sp_remote_tables(
  @server_name
  [, @table_name
  [, @table_owner
  [, @table_qualifier
  [, @with_table_type ] ] ] ]
)
```

Parameters

@server_name

Use this CHAR(128) parameter to specify the remote server name.

@table_name

Use this optional CHAR(128) parameter to specify the name of the remote table. The default is '%'.

@table_owner

Use this optional CHAR(128) parameter to specify the owner of the remote table. The default is '%'.

@table_qualifier

Use this optional CHAR(128) parameter to specify the database in which `table_name` is located. The default is '%'.

@with_table_type

Use this optional BIT parameter to specify the inclusion of remote table types. The default is 0. Specify 1 if you want the result set to include a column that lists table types or specify 0 if you do not.

Result Set

Column name	Data type	Description
database	CHAR(128)	The name of the remote database.
owner	CHAR(128)	The name of the table owner.
table_name	CHAR(128)	The name of the table.
table_type	CHAR(128)	Specifies the table type. The value depends on the type of remote server. For example, TABLE, VIEW, SYS, and GBL TEMP are possible values.

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

It may be helpful when you are configuring your database server to get a list of the remote tables available on a particular server. This procedure returns a list of the tables on a server.

The procedure accepts five parameters. If a table, owner, or database name is given, the list of tables will be limited to only those that match the arguments.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Standards

N/A

Example

The following example returns information about the tables owned by DBA in a SQL Anywhere remote server named RemoteSA.

```
CALL dbo.sp_remote_tables( 'RemoteSA', null, 'DBA' );
```

To get a list of all the tables owned by Fred in the production database in an Adaptive Server Enterprise server named RemoteASE:

```
CALL dbo.sp_remote_tables( 'RemoteASE', null, 'Fred', 'production' );
```

To get a list of all the Microsoft Excel worksheets available from an ODBC data source referenced by a server named RemoteExcel:

```
CALL dbo.sp_remote_tables( 'RemoteExcel' );
```

Use the sp_remote_tables system procedure to see all the subdirectories located in "c:\Program Files" (up to 3 levels deep) on the computer running the database server:

```
CREATE SERVER my_dir_server
CLASS 'DIRECTORY'
USING 'ROOT=c:\Program Files;SUBDIRS=3';
CREATE EXTERNLOGIN DBA TO my_dir_server;
CREATE EXISTING TABLE my_program_files AT 'my_dir_server;;;.';
CALL sp_remote_tables( 'my_dir_server' );
```

Related Information

Remote Data Access
Server Classes for Remote Data Access
CREATE SERVER Statement [page 962]

1.6.8.144 sp_servercaps System Procedure

Displays information about a remote server's capabilities.

Syntax

```
sp_servercaps( @server_name )
```

Parameters

@server_name

Use this CHAR(128) parameter to specify a server defined with the CREATE SERVER statement. `@server_name` is the same server name used in the CREATE SERVER statement.

Results

Column	Type	Description
capid	INTEGER	The capability identifier.
capname	CHAR(128)	The name of the capability.
capvalue	CHAR(128)	The setting of the capability, usually T (true) or F (false).

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

This procedure displays information about a remote server's capabilities. The capability information is used to determine how much of a SQL statement can be forwarded to a remote server. The ISYSCAPABILITY system table, which lists the server capabilities, is not populated until a connection is made to the first remote server.

Standards

N/A

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

To display information about the remote server RemoteSA:

```
CALL dbo.sp_servercaps( 'RemoteSA' );
```

Related Information

Remote Data Access
Server Classes for Remote Data Access
SYSCAPABILITY System View [page 1899]
SYSCAPABILITYNAME System View [page 1900]
CREATE SERVER Statement [page 962]

1.6.8.145 sp_start_listener System Procedure

Starts a new connection listener.

Syntax

```
sp_start_listener(
type
, address
[ , options ]
)
```

Parameters

type

Use this VARCHAR (12) parameter to specify the type of connection listener to start. The value is one of **sharedmemory**, **shmem**, **tcpip**, **tcp**, **http**, or **https**.

address

Use this VARCHAR (100) parameter to specify the address of the connection listener to start. The address is a numeric IP address with a port number (for example, 0.0.0.0:9998) separated by a colon (:) or an IP address without a port number. For IPv6 addresses with a port number, enclose the address in parentheses and then append the colon and port number. If you do not specify a port number, then the default port (TCPIP:2638, HTTP:80, HTTPS:443) is used.

If you specify a port number for TCP/IP and HTTP(S), then the address parameter can be a port number between 1 and 65535. In this case, listeners are started on all available IP addresses using that port number, and the database server acts as though the port number was supplied as the ServerPort (PORT) protocol option to the -x TCPIP or -xs HTTPS(S) database server options.

To indicate all available IPv4 or IPv6 addresses, specify an IP address of "0.0.0.0" or "(::)".

The personal database server only accepts loopback IP addresses, for example 127.0.0.1.

This parameter is ignored for shared memory. For shared memory, specify NULL.

options

Use this LONG VARCHAR parameter to specify a semicolon-delimited list of network protocol options. This parameter is ignored if you are starting shared memory or TCP/IP connection listeners.

> **i** Note
>
> You cannot specify either the ServerPort (PORT) protocol option or the MyIP (ME) protocol option when using the `options` parameter.

Remarks

The new connection listener uses whichever available port number is found first from the following list:

- The port given by the address parameter.
- The default port (2638 for TCP/IP, 80 for HTTP, and 443 for HTTPS).

TCP/IP connection listeners use the encryption setting specified by the -ec database server option when the database server is started.

Shared memory connection listeners can be created regardless of whether or not the -es database server option was specified when the database server was started. Shared memory connection listeners started with the sp_start_listener system procedure always allow unencrypted connections to the database server.

Privileges

You must have the MANAGE LISTENERS system privilege.

Example

Assume that a database server is started allowing local connections only. A problem occurs that is convenient to debug remotely. To allow remote connections to the database server using port 9998, a user connects to the database server by using shared memory and executes the following statement:

```
CALL dbo.sp_start_listener ( 'tcpip' , '0.0.0.0:9998' );
```

Related Information

Network Protocol Options
sp_stop_listener System Procedure [page 1835]

1.6.8.146 sp_stop_listener System Procedure

Stops an existing connection listener.

Syntax

```
sp_stop_listener(
type
, address
[ , force ]
)
```

Parameters

type

Use this VARCHAR (12) parameter to specify the type of connection listener to stop. The value is one of **sharedmemory**, **shmem**, **tcpip**, **tcp**, **http**, **https**.

address

Use this VARCHAR (100) parameter to specify the address of the connection listener to stop. The address is an IP address with a port number separated by a colon (:) or an IP address without a port number. For IPv6 addresses with a port number, enclose the address in parentheses and then append the colon and port number. If a port number is not specified, the default port (TCPIP:2638, HTTP:80, HTTPS:443) is used. For TCP/IP and HTTP(S), the address parameter can be a port number between 1 and 65535. If you only specify a port number, then the database server stops any listeners of the specified type using that port.

To indicate all available IPv4 or IPv6 addresses, specify an IP address of "0.0.0.0" or "(::)".

The personal database server only accepts loopback IP addresses, for example 127.0.0.1.

This parameter is ignored for shared memory. For shared memory, specify NULL.

force

Specify **1** to force the connection listener to stop if it is the last network driver listener running. The default is 0.

Remarks

The sp_stop_listener system procedure only stops new connections from being started on the connection listener. Existing connections are not changed.

Set the `force` parameter to **1** if one of the following is true:

- The connection listener is the last TCP/IP listener and shared memory is not enabled.
- The connection listener is shared memory and there are no TCP/IP listeners running.
- The connection listener is the last HTTP listener and there are no HTTPS listeners running.
- The connection listener is the last HTTPS listener and there are no HTTP listeners running.

Privileges

You must have the MANAGE LISTENERS system privilege.

Example

Assume that a database server is started allowing local connections only. A problem occurs that is convenient to debug remotely. To allow remote connections to the database server using port 9998, a user connects to the database server by using shared memory and executes the following statement:

```
CALL dbo.sp_start_listener ( 'tcpip' , '0.0.0.0:9998' );
```

Once the problem has been solved, shut down the connection listener by executing the following statement:

```
CALL dbo.sp_stop_listener ( 'tcpip' , '0.0.0.0:9998' );
```

Related Information

sp_start_listener System Procedure [page 1833]

1.6.8.147 sp_sys_priv_role_info System Procedure

Returns the one-to-one mapping between system privileges and system roles.

Syntax

sp_sys_priv_role_info( )

Result Set

Column	Type	Description
sys_priv_name	CHAR(128)	The name of the system privilege.
sys_priv_role_name	CHAR(128)	The name of the corresponding system role.
sys_priv_id	UNSIGNED INTEGER	The ID number for the system role.

Remarks

The sp_sys_priv_role_info system procedure reports the complete one-to-one mapping between system privileges and system roles and role IDs. This procedure returns a row for each system privilege.

Designed mainly for internal use, this procedure is used by utilities that need to map a granted system role to the corresponding system privilege and vice versa.

Privileges

You must have EXECUTE privilege on the system procedure.

Standards

ANSI/ISO SQL Standard

Not in the standard.

Example

The following statement returns the mapping of system privileges to their corresponding system role.

```
CALL dbo.sp_sys_priv_role_info();
```

sys_priv_name	sys_priv_role_name	sys_priv_id
VALIDATE ANY OBJECT	SYS_VALIDATE_ANY_OBJECT_ROLE	2147483819
REORGANIZE ANY OBJECT	SYS_REORGANIZE_ANY_OBJECT_ROLE	2147483820
BACKUP DATABASE	SYS_BACKUP_DATABASE_ROLE	2147483821

sys_priv_name	sys_priv_role_name	sys_priv_id
MANAGE ANY EVENT	SYS_MANAGE_ANY_EVENT_ROLE	2147483822
ALTER DATABASE	SYS_ALTER_DATABASE_ROLE	2147483823
SERVER OPERATOR	SYS_SERVER_OPERATOR_ROLE	2147483824
UPGRADE ROLE	SYS_UPGRADE_ROLE_ROLE	2147483825
MANAGE ANY LDAP SERVER	SYS_MAN-AGE_ANY_LDAP_SERVER_ROLE	2147483827
MANAGE CERTIFICATES	SYS_MANAGE_CERTIFICATES_ROLE	2147483828
CREATE ANY SEQUENCE	SYS_CREATE_ANY_SEQUENCE_ROLE	2147483829
...	...	...

Related Information

Roles
sp_has_role System Procedure [page 1782]
sp_proc_priv System Procedure [page 1805]
sp_displayroles System Procedure [page 1768]
sp_objectpermission System Procedure [page 1796]

1.6.8.148 sp_top_k_statements System Procedure

Returns a specified number of statement/plan combinations with the highest maximum runtimes.

Syntax

> **Note**
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Syntax

```
sp_top_k_statements( [ k ] )
```

Parameters

k

(Optional) An UNSIGNED INTEGER that specifies the number of records to return. The default value is 1000.

Result Set

Column name	Data type	Description
stmt_hash	UNSIGNED BIGINT	Returns the statement identifier.
owner_name	CHAR(128)	Returns the name of the owner of the stored procedure. The value is NULL if the statement is not part of a stored procedure, user-defined function, trigger, or event. The value may also be NULL if the statement is executed as part of a stored procedure and the stored procedure is subsequently dropped. A statement executed as part of a stored procedure gets a hash that is related to the procedure, not the statement text.
proc_name	CHAR(128)	Returns the name of the procedure that the statement belongs to. The value is NULL if the statement is not part of a stored procedure, user-defined function, trigger, or event. The value may also be NULL if the statement is executed as part of a stored procedure and the stored procedure is subsequently dropped. A statement executed as part of a stored procedure gets a hash that is related to the procedure, not the statement text.
reusable_stmt_id	UNSIGNED INTEGER	Returns a unique identifier assigned to the statement within a procedure (not necessarily a line number). The value is NULL if the statement is not part of a stored procedure, user-defined function, trigger, or event.
plan_hash	UNSIGNED BIGINT	Returns the plan identifier.
max_seconds	DOUBLE	Returns the maximum runtime observed for the statement when executed with the current plan.
avg_seconds	DOUBLE	Returns the average runtime of the statement with the current plan.

Column name	Data type	Description
stddev_seconds	DOUBLE	Returns the standard deviation of the runtimes of the statement with the current plan. The standard deviation is computed by using the sum of squares method. The value is NULL if only a single execution of a statement has been observed so far.
max_blocking_seconds	DOUBLE	Returns the maximum blocking time observed for the statement with the current plan.
avg_blocking_seconds	DOUBLE	Returns the average blocking time observed for the statement with the current plan.
num_exec	UNSIGNED BIGINT	Returns the number of times the statement was executed using the current plan.
total_num_rows	UNSIGNED BIGINT	Returns the total number of rows returned or modified by the statement over all executions performed with the current plan.
last_max_time _utc	TIMESTAMP	Returns the date and time in Coordinated Universal Time (UTC) that the maximum runtime was last updated.
last_time_utc	TIMESTAMP	Returns the date and time in Coordinated Universal Time (UTC) that the statement statistics were last updated with the current plan.

Remarks

Use this system procedure to determine which statement/plan combination is taking the longest to run. Specify the number of longest running statements returned with the (`k`) parameter.

> **i** Note
>
> If the list of returned statements is long, then space limits may prevent complete data capture.

Privileges

You must have the MONITOR and MANAGE PROFILING privileges on the system procedure.

Side Effects

None.

Example

The following query returns the top statements with the longest maximum observed runtime:

```
SELECT *
FROM dbo.sp_top_k_statements( ) TS
LEFT OUTER JOIN SYS.GTSYSPERFCACHESTMT PS ON TS.stmt_hash = PS.stmt_hash
ORDER BY TS.stmt_hash;
```

Related Information

Tip: Identify the Cause of Slow Statements
sp_find_top_statements System Procedure [page 1773]
GTSYSPERFCACHESTMT System View [page 1897]
GTSYSPERFCACHEPLAN System View [page 1896]

1.6.8.149 sp_trace_event_fields System Procedure

Returns information about the fields of the specified trace event.

Syntax

```
sp_trace_event_fields(
    [ event_name
    [, include_audit_events ] ]
  )
```

Parameters

event_name

Use this optional CHAR(256) parameter to specify the trace event name. The default is NULL.
include_audit_events Use this optional BIT parameter to specify whether or not audit events are returned. This parameter can be 0 (do not return audit events) or 1 (return audit events). By default, audit events are not returned (0 is the default).

Result Set

Column name	Data type	Description
event_name	CHAR(256)	Returns the name of the trace event. System trace event names have the prefix SYS_.
field_name	CHAR(256)	Returns the field name.
field_id	INTEGER	Returns the order of the field in the trace event.
field_domain	INTEGER	Returns the domain ID of the field. Foreign key to SYSDOMAIN.domain_id.
field_description	LONG VARCHAR	Returns a description of the field content.

Remarks

If `event_name` is NULL, this procedure returns the fields for all trace events.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY TRACE SESSION system privilege. You must also have the MANAGE AUDITING system privilege if include_audit_events is not set.

Side Effects

None.

Example

The following statement returns information about the fields for all trace event sessions in the database:

```
SELECT * FROM dbo.sp_trace_event_fields( );
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.6.8.150 sp_trace_event_session_events System Procedure

Lists the trace events that are part of a specific trace event session.

Syntax

```
sp_trace_event_session_events(
    [ event_name
    [, include_server_sessions
    [, include_audit_events ] ] ]
  )
```

Parameters

session_name

Use this optional CHAR(256) parameter to specify the name of the trace event session. The default is NULL. If a session name is not specified or is NULL, information is returned for all trace event sessions.

include_server_sessions Use this optional BIT parameter to specify whether or not engine-level trace event sessions are returned. This parameter can be 0 (do not return engine-level trace event sessions) or 1 (return engine-level trace event sessions). By default, engine-level trace event sessions are not returned (0 is the default).

include_audit_events Use this optional BIT parameter to specify whether or not audit events are returned. This parameter can be 0 (do not return audit events) or 1 (return audit events). By default, audit events are not returned (0 is the default).

Result Set

Column name	Data type	Description
session_name	CHAR(256)	Returns the name of the trace event session.
event_name	CHAR(256)	Returns the trace event name.

Remarks

Use this system procedure to determine which trace events (both system defined and user defined) are being logged for a trace event session.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY TRACE SESSION system privilege. You must also have the MANAGE AUDITING system privilege if include_audit_events is not set.

Side Effects

None.

Example

This statement returns information about the events that are part of all the event tracing sessions for the current database:

```
SELECT * FROM dbo.sp_trace_event_session_events( );
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]

ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.6.8.151 sp_trace_event_session_target_options System Procedure

Lists the target options for a trace event session.

Syntax

```
sp_trace_event_session_target_options(
    [ session_name
    [, include_server_sessions
    [, include_audit_events ] ] ]
  )
```

Parameters

session_name

Use this optional CHAR(256) parameter to specify the name of the trace event session. The default is NULL. If a session name is not specified or is NULL, information is returned for all trace event sessions.

include_server_sessions Use this optional BIT parameter to specify whether or not engine-level trace event sessions are returned. This parameter can be 0 (do not return engine-level trace event sessions) or 1 (return engine-level trace event sessions). By default, engine-level trace event sessions are not returned (0 is the default).

include_audit_events Use this optional BIT parameter to specify whether or not audit events are returned. This parameter can be 0 (do not return audit events) or 1 (return audit events). By default, audit events are not returned (0 is the default).

Result Set

Column name	Data type	Description
session_name	CHAR(256)	Returns the session name.
target_type	CHAR(256)	Returns the target type (such as, FILE).
option_name	CHAR(256)	Returns the option name.
option_value	LONG VARCHAR	Returns the option value.

Remarks

This procedure returns information about the options for one or all trace event sessions for the current database.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY TRACE SESSION system privilege. You must also have the MANAGE AUDITING system privilege if include_audit_events is not set.

Side Effects

None.

Example

The following statement returns information about target options for all trace event sessions in the database:

```
SELECT * FROM dbo.sp_trace_event_session_target_options( );
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]

ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.6.8.152 sp_trace_event_session_targets System Procedure

Lists the targets of a trace session.

Syntax

```
sp_trace_event_session_targets(
    [ session_name
    [, include_server_sessions
    [, include_audit_events ] ] ]
)
```

Parameters

session_name

Use this optional CHAR(256) parameter to specify the name of the trace event session. The default is NULL. If a session name is not specified or is NULL, information is returned about all trace event sessions in the database.

include_server_sessions Use this optional BIT parameter to specify whether or not engine-level trace event sessions are returned. This parameter can be 0 (do not return engine-level trace event sessions) or 1 (return engine-level trace event sessions). By default,engine-level trace event sessions are not returned (0 is the default).

include_audit_events Use this optional BIT parameter to specify whether or not audit events are returned. This parameter can be 0 (do not return audit events) or 1 (return audit events). By default, audit events are not returned (0 is the default).

Result Set

Column name	Data type	Description
session_name	CHAR(256)	Returns the session name.
target_type	CHAR(256)	Returns the target type (for example, a file).

Remarks

A target is the location where the trace event session information is being logged (such as a file or database server message log).

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY TRACE SESSION system privilege. You must also have the MANAGE AUDITING system privilege if include_audit_events is not set.

Side Effects

None.

Example

The following statement returns information about all trace event sessions in the database:

```
SELECT * FROM dbo.sp_trace_event_session_targets( );
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]

DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.6.8.153 sp_trace_event_sessions System Procedure

Returns a list of the trace event sessions that are defined for the database.

Syntax

```
sp_trace_event_sessions( [ session_name ] )
```

Parameters

session_name

Use this CHAR(256) parameter to specify the trace event session you want to get information about. If a session name is not specified, information is returned about all trace event sessions in the database.

Result Set

Column name	Data type	Description
session_name	CHAR(256)	Returns the name of the session.
description	LONG VARCHAR	Returns a description of the session.
started	BIT	Returns 1 if the session has started and 0 otherwise.
is_temporary	BIT	Returns 1 if the trace session is temporary and 0 otherwise.

Remarks

Start and stop trace event sessions manually by using the STATE clause of the ALTER TRACE EVENT SESSION statement.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY TRACE SESSION system privilege.

Side Effects

None

Example

The following statement returns a list of trace event sessions for the current database:

```
SELECT * FROM dbo.sp_trace_event_sessions( );
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_events System Procedure [page 1851]
sp_trace_event_fields System Procedure [page 1841]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.6.8.154 sp_trace_events System Procedure

Returns information about the trace events in the database.

Syntax

```
sp_trace_events(
    [ event_name
    [, include_audit_events ] ]
)
```

Parameters

event_name

Use this optional CHAR(256) parameter to specify the trace event name. The default is NULL.

include_audit_events Use this optional BIT parameter to specify whether or not audit events are returned. This parameter can be 0 (do not return audit events) or 1 (return audit events). By default, audit events are not returned (0 is the default).

Result Set

Column name	Data type	Description
event_name	CHAR(256)	Returns the name of the trace event. System trace event names have the prefix SYS_.
description	LONG VARCHAR	Returns a description of what the trace event captures.
severity	TINYINT	Returns the severity level of the trace event.
is_system	BIT	Returns 1 for system trace events and 0 otherwise.
is_temporary	BIT	Returns 1 for temporary trace events and 0 otherwise.

Remarks

This procedure returns information about both system-defined and user-defined trace events in the database. It returns the details for the specified trace event or for all trace events if `event_name` is not supplied or is NULL.

Level	Severity value range
ALWAYS	0
CRITICAL	1-50
ERROR	51-100
WARNING	101-150
INFORMATION	151-200
DEBUG	201-255

Privileges

You must have EXECUTE privilege on the system procedure, as well as the MANAGE ANY TRACE SESSION system privilege. You must also have the MANAGE AUDITING system privilege if include_audit_events is not set.

Side Effects

None.

Example

The following statement returns a list of user-defined trace events in the database:

```
SELECT * FROM dbo.sp_trace_events( )
WHERE is_system = 0;
```

Related Information

Event Tracing
System Events
CREATE TEMPORARY TRACE EVENT Statement [page 1022]
CREATE TEMPORARY TRACE EVENT SESSION Statement [page 1025]
ALTER TRACE EVENT SESSION Statement [page 765]
DROP TRACE EVENT Statement [page 1139]
DROP TRACE EVENT SESSION Statement [page 1141]
NOTIFY TRACE EVENT Statement [page 1284]
sp_trace_event_fields System Procedure [page 1841]

sp_trace_event_sessions System Procedure [page 1849]
sp_trace_event_session_events System Procedure [page 1843]
sp_trace_event_session_targets System Procedure [page 1847]
sp_trace_event_session_target_options System Procedure [page 1845]
Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.6.8.155 sp_tsql_environment System Procedure

Sets connection options when users connect from jConnect or Open Client applications.

Syntax

```
sp_tsql_environment( )
```

Remarks

The sp_login_environment procedure is the default procedure specified by the login_procedure database option. For each new connection, the procedure specified by login_procedure is called. If the connection uses the TDS communications protocol (that is, if it is an Open Client or jConnect connection), then sp_login_environment in turn calls sp_tsql_environment.

This procedure sets database options so that they are compatible with default Adaptive Server Enterprise behavior.

To change the default behavior, create new procedures and alter your login_procedure option to point to these new procedures.

Below is the list of the options set by sp_tsql_environment procedure:

```
if db_property( 'IQStore' ) = 'Off' then
    -- SQL Anywhere datastore
    SET TEMPORARY OPTION close_on_endtrans='OFF';
end if;
SET TEMPORARY OPTION ansinull='OFF';
SET TEMPORARY OPTION tsql_variables='ON';
SET TEMPORARY OPTION ansi_blanks='ON';
SET TEMPORARY OPTION chained='OFF';
SET TEMPORARY OPTION quoted_identifier='OFF';
SET TEMPORARY OPTION allow_nulls_by_default='OFF';
SET TEMPORARY OPTION on_tsql_error='CONTINUE';
SET TEMPORARY OPTION isolation_level='1';
SET TEMPORARY OPTION date_format='YYYY-MM-DD';
SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
SET TEMPORARY OPTION date_order='MDY';
SET TEMPORARY OPTION escape_character='OFF';
```

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The example below calls the sp_tsql_environment procedure:

```
CALL dbo.sp_tsql_environment();
```

Related Information

sp_login_environment System Procedure [page 1792]
login_procedure Option

1.6.8.156 sp_use_secure_feature_key System Procedure

Enable the features included in the specified secured feature key for the current connection.

Syntax

```
sp_use_secure_feature_key(
name
, auth_key
)
```

Parameters

name

The VARCHAR (128) name of the secured feature key to be enabled.

auth_key

The CHAR (128) case-sensitive authorization code for the secured feature key being enabled. The authorization code must be at least six characters.

Remarks

This procedure enables the secured features that are included in the specified secured feature key for the current connection only.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

In order for the following examples to work, the server must be started with the option: `-sk securefkey`.

The following example enables the SYSTEM secured feature key which includes MANAGE_KEYS for the current connection.

```
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
```

The following example enables the SYSTEM secured feature key which includes MANAGE_KEYS, creates a new secured feature key called MYSET with case-sensitive authorization code SecureMySet, and then uses the new secured feature key:

```
CALL dbo.sp_use_secure_feature_key( 'system', 'securefkey' );
CALL dbo.sp_create_secure_feature_key( 'myset', 'SecureMySet', 'local,remote' );
CALL dbo.sp_use_secure_feature_key( 'myset', 'SecureMySet' );
```

After switching to the new secured feature key, the current connection is no longer using the SYSTEM secured feature so does not have access to MANAGE_KEYS.

Related Information

Creating Secured Feature Keys

sp_alter_secure_feature_key System Procedure [page 1752]
sp_create_secure_feature_key System Procedure [page 1760]
sp_drop_secure_feature_key System Procedure [page 1771]
sp_list_secure_feature_keys System Procedure [page 1791]

1.6.8.157 st_geometry_dump System Procedure

Disassembles a geometry into its lowest level component geometries.

Syntax

```
st_geometry_dump(
geometry
[, "options" ]
)
```

Parameters

geometry

The ST_Geometry value of the geometry to be disassembled.

"options"

A optional VARCHAR(255) string of parameters and values, separated by semicolons, you can use to configure the output of the procedure. The default is NULL.

The following table lists the parameters that can be specified:

Parameter	Default value	Allowed values	Description
Format	*Original*	*Original*, *Internal*, or *Mixed*	The format to return the geometry in. Specifying Original returns the geometry in its original format. Specifying Internal returns the geometry in its normalized format. Specifying Mixed returns whatever stored formats are available, one row per format.

Parameter	Default value	Allowed values	Description
ExpandPoints	*Yes*	*Yes*, *No*	By default, when disassembling a geometry containing points (such as ST_LineString or ST_MultiPoint), the st_geometry_dump system procedure outputs the constituent points to separate rows. Set ExpandPoints to No if you do not want these extra rows to be generated.
MaxDepth	-1	-1, any number greater or equal to zero	By default, st_geometry_dump system procedure continues to disassembles an object hierarchy until it reaches the leaf objects. The MaxDepth parameter can be set to limit the number of levels in the hierarchy the geometry is disassembled. With a value of 0, only the root geometry is returned. With a value of 1, the geometry and its immediate children are returned, and so on.
SetGeom	*Yes*	*Yes, No*	The st_geometry_dump system procedure returns a column that is the ST_Geometry associated with an object in the original type hierarchy. If this column is not needed, the parameter SetGeom can be set to No to reduce the running time and output size of the procedure.

Parameter	Default value	Allowed values	Description
Validate	*Basic*	*None, Basic, Full*	By default, the st_geometry_dump system procedure applies the validation rules that the database server uses when loading geometries, and sets the Valid column of the result set to 1 if the object in the row matches these rules. The Validate parameter can be set to None to disable this checking, or it can be set to Full to also apply the additional checks performed by the ST_IsValid method. Full checking takes longer to perform.

Result Set

Column	Data type	Description
id	UNSIGNED BIGINT	A unique ID for this row in the results.
parent_id	UNSIGNED BIGINT	The ID of the immediate parent of this object.
depth	INTEGER	The depth from the root object to the object associated with this row.
format	VARCHAR(128)	Whether the geometry is the original representation (Original) or the normalized representation (Internal).
valid	BIT	Whether the geometry is valid (1) according to the checking level specified by the Validate option.
geom_type	VARCHAR(128)	The geometry type, as returned by the ST_GeometryType.
geom	ST_Geometry	The geometry specification. If SetGeom parameter is set to No, the geometry specification is not returned in the result set.
xmin	DOUBLE	The minimum x value for the geometry.
xmax	DOUBLE	The maximum x value for the geometry.
ymin	DOUBLE	The minimum y value for the geometry.
ymax	DOUBLE	The maximum y value for the geometry.
zmin	DOUBLE	The minimum z value for the geometry.

Column	Data type	Description
zmax	DOUBLE	The maximum z value for the geometry.
mmin	DOUBLE	The minimum m value for the geometry.
mmax	DOUBLE	The maximum m value for the geometry.
details	LONG VARCHAR	Any extra details about the geometry, including additional information about why the object is not valid.

Remarks

The st_geometry_dump system procedure disassembles a geometry hierarchy with one row for each of the objects in the hierarchy (including the root object). Each geometry in the hierarchy can be validated to find out if it is valid, and if not, why.

Some of the functionality of the st_geometry_dump system procedure can be matched by using type-specific methods such as ST_GeometryN or ST_PointN.

The st_geometry_dump system procedure can be used to correct invalid geometries.

Privileges

You must have EXECUTE privilege on the system procedure.

Side Effects

None

Example

The following example disassembles the polygon, `'Polygon ((0 0, 3 0, 3 3, 0 3, 0 0))'`, into its component geometries:

```
SELECT * FROM dbo.st_geometry_dump( 'Polygon ((0 0, 3 0, 3 3, 0 3, 0 0))',
'SetGeom=No' );
```

id	parent_id	depth	format	valid	geom_type	geom	xmin	xmax	ymin	ymax	...
1	1	0	Internal	1	ST_Polygon	Polygon ((0 0, 3 0, 3 3, 0 3, 0 0))	0	3	0	3	...
2	1	1	Internal	1	ST_LineString	LineString (0 0, 3 0, 3 3, 0 3, 0 0)	0	3	0	3	...
3	2	2	Internal	1	ST_Point	Point (0 0)	0	0	0	0	...
4	2	2	Internal	1	ST_Point	Point (3 0)	3	3	0	0	...
5	2	2	Internal	1	ST_Point	Point (3 3)	3	3	3	3	...
6	2	2	Internal	1	ST_Point	Point (0 3)	0	0	3	3	...
7	2	2	Internal	1	ST_Point	Point (0 0)	0	0	0	0	...

The following example shows how the st_geometry_dump system procedure can be used to find the invalid points within a geometry. In this example, the linestring contains a point with longitude 1200. Because of this, the point and the linestring are both reported as invalid (valid=0) in the results.

```
SET TEMPORARY OPTION st_geometry_on_invalid='Ignore';
CREATE OR REPLACE VARIABLE @geo ST_Geometry;
SET @geo = new ST_LineString( 'LineString(1200 2, 80 10)', 4326 );
SELECT * FROM dbo.st_geometry_dump( @geo, 'SetGeom=No' );
```

id	parent_id	depth	format	valid	geom_type	geom	xmin	xmax	ymin	ymax	...	details
1	1	0	Original	0	ST_LineString	(NULL)	80	1,200	2	10	...	Value 1200.000000 out of range for coordinate longitude (SRS allows -180.000000 to 180.000000).
2	1	1	Original	0	ST_LineString	(NULL)	1,200	1,200	2	2	...	Value 1200.000000 out of range for coordinate longitude (SRS allows -180.000000 to 180.000000).
3	1	1	Original	1	ST_Point	(NULL)	80	80	10	10	...	

Once invalid data has been identified, the st_geometry_dump system procedure can be used with other spatial methods to correct the invalid elements to assemble a valid geometry. The following example shows how an invalid point with longitude 1200 can be corrected to have longitude 120.0:

```
SET TEMPORARY OPTION st_geometry_on_invalid='Ignore';
CREATE OR REPLACE VARIABLE @geo ST_Geometry;
SET @geo = new ST_LineString( 'LineString(1200 2, 80 10)', 4326 );
SELECT ST_LineString::ST_LineStringAggr(
    new ST_Point( IF xmax = 1200 then 120.0 ELSE xmax ENDIF,
                  ymax, 4326 )  ORDER BY id )
FROM  dbo.st_geometry_dump( @geo )
WHERE geom_type='ST_Point';
```

Related Information

ST_IsValid Method
st_geometry_on_invalid Option

1.6.8.158 st_geometry_load_shapefile System Procedure

Creates a table and loads an ESRI shapefile into it.

Syntax

```
st_geometry_load_shapefile(
shp_filename
, srid
, table_name
[, table_owner
[, shp_encoding ] ]
)
```

Parameters

shp_filename

Use this VARCHAR(512) parameter to specify the location of the ESRI shapefile. The file name must have the extension `.shp` and must have associated `.shx` and `.dbf` files with the same base name located in the same directory. The path is relative to the database server, not the client application.

srid

Use this INTEGER parameter to specify the SRID to associate with the data from the shapefile. For the geometries to be meaningful, the SRID you specify should be appropriate for the geometries, even if it isn't identical to the SRID used in the shapefile.

table_name

Use this VARCHAR(128) column to specify the name of the table to create to hold the shapefile data.

table_owner

Use this optional VARCHAR(128) column to specify the owner for the new table that is created. The default owner is the current user.

shp_encoding

Use this optional VARCHAR(50) parameter to specify the encoding to use when reading the shapefile. The default encoding is ISO-8859-1.

Remarks

The st_geometry_load_shapefile system procedure first creates the table "table_owner"."table_name" using the column information (names and types) found in the ESRI shapefile. st_geometry_load_shapefile then loads the data from the shapefile into the new table.

This procedures makes use of the sa_describe_shapefile system procedure, and the LOAD TABLE...FORMAT SHAPEFILE statement.

Privileges

You must have EXECUTE privilege on the system procedure, as well as other privileges, as indicated below:

If the -gl option is set to DBA:

- If you are `table_owner`, you must have one of CREATE TABLE, CREATE ANY TABLE, or CREATE ANY OBJECT system privilege, and you must have one of ALTER ANY TABLE, ALTER ANY OBJECT, or LOAD ANY TABLE system privilege.
- If you are not `table_owner`, you must have one of CREATE ANY TABLE or CREATE ANY OBJECT system privilege, and you must have one of ALTER ANY TABLE, ALTER ANY OBJECT, or LOAD ANY TABLE system privilege.

If the -gl option is set to ALL:

- If you are `table_owner`, you must have one of CREATE TABLE, CREATE ANY TABLE, or CREATE ANY OBJECT system privilege.
- If you are not `table_owner`, you must have one of CREATE ANY TABLE or CREATE ANY OBJECT system privilege, and you must have one of ALTER ANY TABLE, ALTER ANY OBJECT, or LOAD ANY TABLE system privilege.

If the -gl option is set to NONE, loading the data fails.

Side Effects

If loading fails, the created table remains.

Example

The following statement creates the table esri_load and loads it with the data from a fictitious shapefile `c:\esri\shapefile.shp`, associating the data with SRID 1000004326:

```
CALL dbo.st_geometry_load_shapefile( 'c:\\esri\\tgr36069trt00.shp', 1000004326,
'esri_load' );
```

Related Information

Lesson 3: Load the ESRI Shapefile Data
sa_describe_shapefile System Procedure [page 1580]
LOAD TABLE Statement [page 1247]
Support for ESRI Shapefiles

1.6.8.159 xp_cmdshell System Procedure

Carries out an operating system command from a procedure.

Syntax

```
xp_cmdshell(
command
[, redir_output | 'no_output' ]
)
```

Parameters

command

Use this VARCHAR(8000) parameter to specify a system command. The default is NULL.

redir_output

Use this optional CHAR(254) parameter to specify whether to display output in a command window. The default behavior is to display output in a command window. If you specify 'no_output', output is not displayed in a command window. The default value is ' '.

Returns

This function returns an INTEGER exit code.

Remarks

xp_cmdshell executes a system command and then returns control to the calling environment. The value returned by xp_cmdshell is the exit code from the executed shell process. The return value is 2 if an error occurs when the child process is started.

The second parameter affects only command line applications on Windows operating systems. For UNIX/Linux, no command window appears, regardless of the setting for the second parameter.

Use the sa_enable_auditing_type and sa_disable_auditing_type system procedures to enable and disable auditing of the xp_cmdshell system procedure (using the xp_cmdshell type). When auditing is enabled for xp_cmdshell, all invocations of the xp_cmdshell procedure are logged to the audit log, including the user that executed the procedure, and the parameters that were passed to the procedure.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

Example

The following statement lists the files in the current directory in the file `c:\temp.txt`:

```
CALL dbo.xp_cmdshell( 'dir > c:\\temp.txt' );
```

The following statement carries out the same operation, but does so without displaying a *Command* window.

```
CALL dbo.xp_cmdshell( 'dir > c:\\temp.txt', 'no_output' );
```

Related Information

General Security Tips
CALL Statement [page 795]
Directory and File System Procedures [page 1516]

1.6.8.160 xp_get_mail_error_code System Procedure

Returns information about the most recent SMTP or MAPI error.

Syntax

xp_get_mail_error_code( )

Returns

This function returns an INTEGER value representing the SMTP or MAPI error code.

Remarks

When the return value of a mail procedure (xp_startmail, xp_startsmtp, xp_sendmail, xp_stopmail, and xp_stopsmtp) is -1, use this function to retrieve the SMTP or MAPI error code.

When the return value of a mail procedure is 5, 6, or 7, use this function to retrieve the error number for the most recent socket error.

If MAPI is being used, the value returned is the return code of the MAPI function. If SMTP is being used, the value returned is either an SMTP error code (and xp_get_mail_error_text returns the SMTP error text) or an error number (and xp_get_mail_error_text returns an empty string).

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SEND EMAIL system privilege.

Side Effects

None

Example

This example gets the most recent SMTP or MAPI error code.

```
SELECT dbo.xp_get_mail_error_code( )
```

This example uses SMTP to initiate the sending of a plain text message.

```
BEGIN
    DECLARE err_smtp INTEGER;
    DECLARE err_code INTEGER;
    DECLARE err_msg  LONG VARCHAR;
    SELECT dbo.xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' ) INTO
err_smtp;
    SELECT dbo.xp_get_mail_error_code( ), xp_get_mail_error_text( ) INTO
err_code, err_msg;
    SELECT err_smtp, err_code, err_msg;
END;
```

Related Information

Status Codes for MAPI and SMTP System Procedures [page 1515]
xp_get_mail_error_text System Procedure [page 1867]

xp_startmail System Procedure [page 1880]
xp_startsmtp System Procedure [page 1881]
xp_sendmail System Procedure [page 1874]
xp_stopmail System Procedure [page 1885]
xp_stopsmtp System Procedure [page 1886]

1.6.8.161 xp_get_mail_error_text System Procedure

Returns the most recent SMTP error or status message text.

Syntax

```
xp_get_mail_error_text( )
```

Return value

This function returns a LONG VARCHAR value representing the SMTP or MAPI error or status message text. If no error text is available, an empty string or NULL is returned.

Remarks

Use this function to obtain the error or status message text for any of the mail procedures (xp_startmail, xp_startsmtp, xp_sendmail, xp_stopmail, and xp_stopsmtp).

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SEND EMAIL system privilege.

Side Effects

None

Example

This example gets the most recent SMTP or MAPI message text.

```
SELECT xp_get_mail_error_text( )
```

This example uses SMTP to initiate the sending of a plain text message.

```
BEGIN
    DECLARE err_smtp INTEGER;
    DECLARE err_code INTEGER;
    DECLARE err_msg  LONG VARCHAR;
    SELECT dbo.xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' ) INTO
err_smtp;
    SELECT dbo.xp_get_mail_error_code( ), xp_get_mail_error_text( ) INTO
err_code, err_msg;
    SELECT err_smtp, err_code, err_msg;
END;
```

Related Information

Status Codes for MAPI and SMTP System Procedures [page 1515]
xp_get_mail_error_code System Procedure [page 1865]
xp_startmail System Procedure [page 1880]
xp_startsmtp System Procedure [page 1881]
xp_sendmail System Procedure [page 1874]
xp_stopmail System Procedure [page 1885]
xp_stopsmtp System Procedure [page 1886]

1.6.8.162 xp_getenv System Procedure

Returns the value of an environment variable.

Syntax

```
xp_getenv( environment_variable )
```

Parameters

environment_variable

Use this VARCHAR(8000) parameter to specify the environment variable. This parameter is case insensitive on Windows operating systems and case sensitive on all other operating systems, independent of the case sensitivity of the database. The default value is NULL.

Returns

This function returns a LONG NVARCHAR value.

Remarks

If the environment variable specified is NULL or not set, NULL is returned.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SERVER OPERATOR system privilege.

The GETENV feature must be enabled for the connection (-sf server option).

Side Effects

None

Example

The following example uses the xp_getenv system procedure to return the value of the environment variable PATH.

```
SELECT CAST( dbo.xp_getenv( 'PATH' ) AS LONG VARCHAR );
```

The following example uses the xp_getenv and sa_split_list system procedures to return the value of the Windows environment variable PATH as a list. Use ':' as the separator character on UNIX and Linux operating systems.

```
CALL sa_split_list( CAST( dbo.xp_getenv( 'PATH' ) AS LONG VARCHAR ), ';' );
```

The following example uses the environment variable NONEXISTENT, which is assumed to not exist. Therefore, the query is expected to return NULL.

```
SELECT dbo.xp_getenv( 'NONEXISTENT' );
```

Related Information

Environment Variables

1.6.8.163 xp_msver System Procedure

Retrieves version and name information about the database server.

Syntax

```
xp_msver( the_option )
```

Parameters

the_option

Use this CHAR(254) parameter to specify a string. The string must be one of the following, enclosed in string delimiters.

Argument	Description
ProductName	Returns the name of the product. ProductName is the default argument value.
ProductVersion	Returns the version number, followed by the build number. The format is as follows: `17.0.11.1293`
CompanyName	Returns the name of the company.
FileDescription	Returns the name of the product, followed by the name of the operating system.
LegalCopyright	Returns a copyright string for the software.
LegalTrademarks	Returns trademark information for the software.

Returns

This function returns a CHAR(254) value.

Remarks

This functions returns product, company, version, and other information.

Privileges

You must have EXECUTE privilege on the system procedure.

Example

The following statement requests the version and operating system description:

```
SELECT dbo.xp_msver( 'ProductVersion') Version,
   xp_msver( 'FileDescription' ) Description;
```

Sample output is as follows. The value for Version will likely be different on your system.

Version	Description
17.0.11.1293	SQL Anywhere Windows7

Related Information

System Functions [page 224]

1.6.8.164 xp_read_file System Procedure

Reads a file and returns the contents of the file as a LONG BINARY variable.

Syntax

```
xp_read_file(
filename
[, lazy ]
)
```

Parameters

filename

Use this LONG VARCHAR parameter to specify the name of the file for which to return the contents.

lazy

When you specify this optional INTEGER parameter and its value is not 0, the contents of the file are not read until they are requested. Reads only occur when the LONG BINARY value is accessed and only on the portion of the file that is requested. The default is 0, or non-lazy.

Returns

This function returns the contents of the named file as a LONG BINARY value. If the file does not exist or cannot be read, NULL is returned.

Remarks

The `filename` is relative to the starting directory of the database server.

The function can be useful for inserting entire documents or images stored in files into tables. If the file cannot be read, the function returns NULL.

If the data file is in a different character set, you can use the CSCONVERT function to convert it.

You can also use the CSCONVERT function to address character set conversion requirements you have when using the xp_read_file system procedure.

If disk sandboxing is enabled, the file referenced in `filename` must be in an accessible location.

The function returns NULL if the specified file does not exist.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the READ FILE system privilege.

Example

The following statement inserts an image into a column named Photo of the Products table.

```
UPDATE Products
SET Photo=dbo.xp_read_file( 'c:\\sqlany\\scripts\\adata\\HoodedSweatshirt.jpg' )
WHERE Products.ID=600;
```

The following statement reads a text file and displays each line with a line number.

```
SELECT * FROM sa_split_list( CAST( dbo.xp_read_file( '\\Windows\\win.ini' ) AS
LONG VARCHAR ), 0x0a );
```

Related Information

XML Import Using the OPENXML Operator
Insertion of Documents and Images
Disk Sandboxing
CSCONVERT Function [String] [page 309]
xp_write_file System Procedure [page 1887]
Directory and File System Procedures [page 1516]
CALL Statement [page 795]

1.6.8.165 xp_scanf System Procedure

Extracts substrings from an input string using a format string.

Syntax

```
xp_scanf(
input_buffer
, format
[ , param1 [, param2 ... ] ]
)
```

Parameters

input_buffer

Use this CHAR(254) parameter to specify the input string.

format

Use this CHAR(254) parameter to specify the format of the input string, using place holders (%s) for each `param` argument. There can be up to fifty place holders in the `format` argument, and there must be the same number of place holders as `param` arguments.

param1, param2, ...

Use one or more of these CHAR(254) parameters to store the substrings extracted from `input_buffer`. There can be up to 50 of these parameters.

Privileges

You must have EXECUTE privilege on the system procedure.

Remarks

The xp_scanf system procedure extracts substrings from an input string using the specified format, and puts the results in the specified parameter values.

Only the %s string format is supported. Other format specifiers such as %d and %f are not supported and scanning the input string stops if they are encountered.

Example

The following statements extract the substrings Hello and World! from the input buffer Hello World!, and puts them into variables string1 and string2, and then selects them:

```
CREATE VARIABLE string1 CHAR( 254 );
CREATE VARIABLE string2 CHAR( 254 );
CALL dbo.xp_scanf( 'Hello World!', '%s %s', string1, string2 );
SELECT string1, string2;
```

The following statements show how to take a date string and split it into its year, month, and day components:

```
CREATE VARIABLE ymd LONG VARCHAR;
CREATE VARIABLE year CHAR( 254 );
CREATE VARIABLE month CHAR( 254 );
CREATE VARIABLE day CHAR( 254 );
SET ymd = '2014/11/23';
SET ymd = REPLACE(ymd, '/', ' ');
CALL dbo.xp_scanf( ymd, '%s %s %s', year, month, day );
SELECT ymd, year, month, day;
```

Related Information

CALL Statement [page 795]

1.6.8.166 xp_sendmail System Procedure

Sends an email message to the specified recipients once a session has been started with xp_startmail or xp_startsmtp. The procedure accepts messages of any length.

Syntax

```
xp_sendmail(
  recipient = mail-address
  [, subject = subject ]
  [, cc_recipient = mail-address ]
  [, bcc_recipient = mail-address ]
  [, query = sql-query ]
  [, "message" = message-body ]
```

```
  [, attachname = attach-name ]
  [, attach_result = attach-result ]
  [, echo_error = echo-error ]
  [, include_file = filename ]
  [, no_column_header = no-column-header ]
  [, no_output = no-output ]
  [, width = width ]
  [, separator = separator-char ]
  [, dbuser = user-name ]
  [, dbname = db-name ]
  [, type = type ]
  [, include_query = include-query ]
  [, content_type = content-type ]
)
```

Parameters

Some arguments supply fixed values and are available for use to ensure Transact-SQL compatibility, as noted below.

recipient

This LONG VARCHAR parameter specifies the recipient mail address. When specifying multiple recipients, each mail address must be separated by a semicolon.

subject

This LONG VARCHAR parameter specifies the subject field of the message. The default is NULL.

cc_recipient

This LONG VARCHAR parameter specifies the cc recipient mail address. When specifying multiple cc recipients, each mail address must be separated by a semicolon. The default is NULL.

bcc_recipient

This LONG VARCHAR parameter specifies the bcc recipient mail address. When specifying multiple bcc recipients, each mail address must be separated by a semicolon. The default is NULL.

query

This LONG VARCHAR is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is NULL.

"message"

This LONG VARCHAR parameter specifies the message contents. The default is NULL. The "message" parameter name requires double quotes around it because MESSAGE is a reserved word.

attachname

This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is NULL.

attach_result

This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

echo_error

This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 1.

include_file

This LONG VARCHAR parameter specifies an attachment file. The default is NULL.

no_column_header

This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

no_output

This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

width

This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 80.

separator

This CHAR(1) parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is CHAR(9).

dbuser

This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is guest.

dbname

This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is master.

type

This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is NULL.

include_query

This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

content_type

This LONG VARCHAR parameter specifies the content type for the "message" parameter (for example, text/html, ASIS, and so on). The default is NULL. The value of content_type is not validated; setting an invalid content type results in an invalid or incomprehensible email being sent.

To set headers manually, set the content_type parameter to ASIS. When you do this, the xp_sendmail procedure assumes that the data passed to the message parameter is a properly formed email with headers, and does not add any additional headers. When specifying ASIS, you must set all the headers manually in the message parameter, even headers that would normally be filled in by passing data to the other parameters.

Returns

This function returns an INTEGER status code.

Remarks

The argument values for xp_sendmail are strings. The length of each argument is limited to the amount of available memory on your system.

The content_type argument is intended for users who understand the requirements of MIME email. xp_sendmail accepts ASIS as a content_type. When content_type is set to ASIS, xp_sendmail assumes that the message body ("message") is a properly formed email with headers, and does not add any additional headers. Specify ASIS to send multipart messages containing more than one content type.

Any attachment specified by the include_file parameter is sent as application/octet-stream MIME type, with base64 encoding, and must be present on the database server.

Email sent with an SMTP email system is encoded if the subject line contains characters that are not 7-bit ASCII. Also, email sent to an SMS-capable device may not be decoded properly if the subject line contains characters that are not 7-bit ASCII.

You must have executed xp_startmail to start an email session using MAPI, or xp_startsmtp to start an email session using SMTP.

If you are sending mail using MAPI, the content_type parameter is not supported.

If `message-body` contains lines that are longer than 998 characters, the SMTP server may insert newline characters as well as ! characters into the body of the email. To avoid these extra characters, ensure `message-body` does not contain lines longer than 998 characters.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SEND EMAIL system privilege.

Example

This example uses SMTP to send a plain text message.

```
CALL dbo.xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL dbo.xp_sendmail( recipient='jane.smith@sample.com',
                  subject='This is my subject line',
                  "message"='This text is the body of my email.\n' );
CALL dbo.xp_stopsmtp( );
```

This example uses SMTP to send an HTML formatted message with an attachment.

```
CALL dbo.xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL dbo.xp_sendmail( recipient='jane.smith@sample.com',
                  subject='HTML mail example with attachment',
                  "message"='Plain text.<BR><BR><B>Bold text.</B><BR><BR><BR>' ||
                            '<a href="www.sap.com">SAP Home Page</a>',
                  content_type = 'text/html',
                  include_file = '\\temp\\sendmail2.sql' );
CALL dbo.xp_stopsmtp( );
```

This example uses SMTP to send an inline HTML formatted message with an attachment.

```
CALL dbo.xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL dbo.xp_sendmail( recipient='jane.smith@sample.com',
                  subject='Inline HTML mail example with attachment',
                  "message"='Content-Type: text/html;\nContent-Disposition:
inline; \n\n' ||
                            'Plain text.<BR><BR><B>Bold text.</B><BR><BR>' ||
                            '<a href="www.sap.com">SAP Home Page</a>',
                  content_type = 'ASIS',
                  include_file = '\\temp\\sendmail3.sql' );
CALL dbo.xp_stopsmtp( );
```

This example uses SMTP to send an inline HTML formatted message with a signature and two attachments, one of which is a ZIP file.

```
BEGIN
DECLARE content LONG VARCHAR;
SET content =
'Content-Type: multipart/mixed; boundary="xxxxx";\n' ||
'This part of the email should not be shown.  If this ' ||
'is shown then the email client is not MIME compatible\n\n' ||
'--xxxxx\n' ||
'Content-Type: text/html;\n' ||
'Content-Disposition: inline;\n\n' ||
'Plain text.<BR><BR><B>Bold text.</B><BR><BR>' ||
'<a href="www.sap.com">SAP Home Page</a>\n\n' ||
xp_read_file( '\\temp\\johndoe.sig.html' ) ||
'--xxxxx\n' ||
'Content-Type: application/zip; name="sendmail4.zip"\n' ||
'Content-Transfer-Encoding: base64\n' ||
'Content-Disposition: attachment; filename="sendmail4.zip"\n\n' ||
base64_encode( xp_read_file( '\\temp\\sendmail4.zip' ) ) ||
'\n\n' ||
'--xxxxx--\n';
CALL dbo.xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL dbo.xp_sendmail( recipient='jane.smith@sample.com',
                  subject='Inline HTML mail example with signature and 2
attachments',
                  "message"=content,
                  content_type = 'ASIS',
                  include_file = '\\temp\\sendmail4.sql' );
CALL dbo.xp_stopsmtp( );
END
```

Related Information

MAPI and SMTP System Procedures [page 1513]
Reserved Words [page 6]
Status Codes for MAPI and SMTP System Procedures [page 1515]
xp_startmail System Procedure [page 1880]
xp_startsmtp System Procedure [page 1881]
xp_stopmail System Procedure [page 1885]
xp_stopsmtp System Procedure [page 1886]
CALL Statement [page 795]

1.6.8.167 xp_sprintf System Procedure

Builds a result string from a set of input strings.

Syntax

```
xp_sprintf(
buffer
, format
[ , param1 [, param2 ... ] ]
)
```

Parameters

buffer

This is a CHAR(254) OUT parameter that is filled in with the formatted result.

format

Use this CHAR(254) parameter to specify how to format the result string, using place holders (%s) for each `param` argument. There can be up to fifty place holders in the `format` argument, and there should be the same number of place holders as `param` arguments. Only the %s string format is supported.

param1, param2

The input strings that are used in the result string. You can specify up to 50 of these CHAR(254) arguments.

Remarks

The result placed in the output parameter is truncated to 254 characters.

Privileges

You must have the EXECUTE privilege on the system procedure.

Example

The following statements put the string Hello World! into the result variable.

```
CREATE VARIABLE result CHAR( 254 );
CALL dbo.xp_sprintf( result, '%s %s', 'Hello', 'World!' );
SELECT result;
```

The following statements format the year, month, and day into a date string.

```
CREATE VARIABLE result CHAR( 254 );
CALL dbo.xp_sprintf( result, '%s/%s/%s', 2014, 11, 23 );
SELECT result;
```

Related Information

CALL Statement [page 795]

1.6.8.168 xp_startmail System Procedure

Starts an email session under MAPI.

Syntax

```
xp_startmail(
[ mail_user = mail-login-name
[, mail_password = mail-password ]  ]
)
```

Parameters

mail_user

Use this LONG VARCHAR parameter to specify the MAPI login name. The default is NULL.

mail_password

Use this LONG VARCHAR parameter to specify the MAPI password. The default is NULL.

Returns

This function returns an INTEGER status code.

Remarks

xp_startmail is a system procedure owned by `dbo` that starts an email session.

If you are using Microsoft Exchange, the `mail-login-name` argument is an Exchange profile name, and you should not include a password in the procedure call.

Not supported on UNIX and Linux.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SEND EMAIL system privilege.

Related Information

MAPI and SMTP System Procedures [page 1513]
General Security Tips
Status Codes for MAPI and SMTP System Procedures [page 1515]
xp_stopmail System Procedure [page 1885]
xp_sendmail System Procedure [page 1874]
xp_startsmtp System Procedure [page 1881]
xp_stopsmtp System Procedure [page 1886]
CALL Statement [page 795]

1.6.8.169 xp_startsmtp System Procedure

Starts an email session under SMTP.

Syntax

```
xp_startsmtp(
smtp_sender = email-address
, smtp_server = smtp-server
 [, smtp_port = port-number ]
 [, timeout = timeout ]
 [, smtp_sender_name = username ]
 [, smtp_auth_username = auth-username ]
 [, smtp_auth_password = auth-password ]
 [, trusted_certificates = { public-certificate | * }
 [, secure = { 1 | 0 } ]
 [, certificate_company = organization ]
 [, certificate_unit = organization-unit ]
 [, certificate_name = common-name ]
 [, skip_certificate_name_check= { 1 | 0 } ]
)
```

Parameters

smtp_sender

This LONG VARCHAR parameter specifies the email address of the sender.

smtp_server

This LONG VARCHAR parameter specifies which SMTP server to use and is comprised of the SMTP server name or IP address.

smtp_port

This optional INTEGER parameter specifies the port number to connect to on the SMTP server. The default is 25.

timeout

This optional INTEGER parameter specifies how long to wait, in seconds, for a response from the database server before aborting the current call to xp_sendmail. The default is 60 seconds.

smtp_sender_name

This optional LONG VARCHAR parameter specifies an alias for the sender's email address. For example, JSmith instead of `email-address`. The default is NULL.

smtp_auth_username

This optional LONG VARCHAR parameter specifies the user name to provide to SMTP servers requiring authentication. The default is NULL.

smtp_auth_password

This optional LONG VARCHAR parameter specifies the password to provide to SMTP servers requiring authentication. The default is NULL.

trusted_certificates

This optional LONG VARCHAR parameter is a list of keyword=value pairs separated by semicolons. The default is NULL. When this parameter is NULL, a standard SMTP connection is made. The possible keys are listed below. Only one of the file, certificate, and cert_name options should be specified.

This parameter takes the filename given by FILE=key and contains a list of PEM-encoded X.509 trusted root certificates.

The trusted certificate can be a server's self-signed certificate, a public root certificate, or a certificate belonging to a commercial Certificate Authority. Generate your certificates using RSA.

To use a certificate from the operating system's certificate store, specify **`file=*`**.

To make secure SMTP (SMTPS) connections, which use TLS authentication and encryption, specify **`SMTPS=YES`**.

A single file name can also be specified (**`trusted_certificates=`**`file-spec`).

The secure and trusted_certificates options can be used together to indicate how to connect to the server. The following table describes the different possibilities.

Key	Value
file=	The path and file name of a file that contains one or more trusted certificates.
cert_name=	The name of a certificate stored in the database.
certificate=	The certificate data.
SMTPS=	*YES* | *NO*

secure

This optional parameter specifies whether the connection is secure and whether to use a specified trusted certificate or a certificate from the operating system's certificate store. The default is NULL.

	secure=NULL	secure=0	secure=1
trusted_certificate=NULL	Not secure	Not secure	Secure. Uses operating system certificate store.
trusted_certificate='*'	Secure. Uses operating system certificate store.	Returns an error.	Secure. Uses operating system certificate store.
trusted_certificate=`filename`	Secure. Uses specified certificate.	Returns an error.	Secure. Uses specified certificate.

> **i** Note
>
> Specify *trusted_certificate=none* to make a TLS connection without verifying the server's certificate. Connecting without verifying the server's certificate is less secure than verifying the certificate because the client can no longer protect against a man-in-the-middle attack. However, the connection is still strongly encrypted and prevents replay attacks, which is more secure than no encryption at all. With the *none* option, no trusted root certificate is required on the client.

certificate_company

This optional LONG VARCHAR parameter specifies that the client accepts server certificates only when the Organization field of the certificate matches this value. This parameter is ignored when the trusted_certificates value is NULL. The default is NULL.

certificate_unit

This optional LONG VARCHAR parameter specifies that the client accepts server certificates only when the Organization Unit field of the certificate matches this value.

certificate_name

This optional LONG VARCHAR parameter specifies that the client accepts server certificates only when the Common Name field on the certificate matches this value. This parameter is ignored when the trusted_certificates value is NULL. The default is NULL.

skip_certificate_name_check

This optional BIT parameter controls whether the SMTP server's host is checked against the SMTP server certificate. Specifying 1 enables this option. The default is 0. This parameter is ignored when the trusted_certificates value is NULL, or when any of the following parameters are specified: certificate_company, certificate_unit, or certificate_name.

> **i** Note
>
> Setting this parameter to 1 is not recommended because this setting prevents the database server from fully authenticating the SMTP server.

Returns

This function returns an INTEGER status code (SMTP/MAPI return code).

Remarks

xp_startsmtp is a system procedure that starts a mail session for a specified email address by connecting to an SMTP server. This connection can time out. You should call xp_startsmtp just before executing xp_sendmail.

The database server supports CRAM-MD5 authentication, as well as PLAIN authentication. When you use the xp_startsmtp system procedure with the smtp_auth_username and smtp_auth_password parameters, the database server uses CRAM-MD5 authentication. If the SMTP server does not support CRAM-MD5 authentication, then the database server uses PLAIN authentication. If the SMTP server does not support the SMTP authentication capability, error code 104 (Server error; response not understood) is returned. Also, if the database server is started using the -fips database server option, then only PLAIN authentication is used.

CRAM-MD5 authentication is more secure than PLAIN authentication, but neither encrypts what is sent to the SMTP server. To encrypt what is sent to the SMTP server, including email messages, use secure SMTP. Secure SMTP uses TLS encryption to encrypt and can be used with CRAM-MD5 or PLAIN authentication.

Virus scanners can affect xp_startsmtp, causing it to return error code 100. For McAfee VirusScan version 8.0.0 and later, settings for preventing mass mailing of email worms also prevent xp_sendmail from executing properly. If your virus scanning software allows you to specify processes that can bypass the mass mailing protections, specify *dbeng17*`.exe` and *dbsrv17*`.exe`. For example, with McAfee VirusScan you can allow mass mailing for these two processes by adding them to the list of *Excluded Processes* in the *Properties* area.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SEND EMAIL system privilege.

Related Information

MAPI and SMTP System Procedures [page 1513]
General Security Tips
Digital Certificates
Status Codes for MAPI and SMTP System Procedures [page 1515]
xp_startmail System Procedure [page 1880]
xp_stopmail System Procedure [page 1885]
xp_sendmail System Procedure [page 1874]
xp_stopsmtp System Procedure [page 1886]
CALL Statement [page 795]
CREATE CERTIFICATE Statement [page 820]
trusted_certificates_file Option
Encryption (ENC) Connection Parameter

1.6.8.170 xp_stopmail System Procedure

Closes a MAPI email session.

Syntax

```
xp_stopmail( )
```

Returns

This function returns an INTEGER status code.

Remarks

xp_stopmail is a system procedure that ends an email session.

Not supported on UNIX and Linux.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SEND EMAIL system privilege.

Example

The following statement ends the email session.

```
CALL dbo.xp_stopmail( );
```

Related Information

MAPI and SMTP System Procedures [page 1513]
General Security Tips
Status Codes for MAPI and SMTP System Procedures [page 1515]
xp_startmail System Procedure [page 1880]
xp_sendmail System Procedure [page 1874]
xp_startsmtp System Procedure [page 1881]

xp_stopsmtp System Procedure [page 1886]
CALL Statement [page 795]

1.6.8.171 xp_stopsmtp System Procedure

Closes an SMTP email session.

Syntax

```
xp_stopsmtp( )
```

Returns

This function returns an INTEGER status code.

Remarks

xp_stopsmtp is a system procedure that ends an email session.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the SEND EMAIL system privilege.

Example

The following statement ends the email session.

```
CALL dbo.xp_stopsmtp( );
```

Related Information

MAPI and SMTP System Procedures [page 1513]
General Security Tips
Status Codes for MAPI and SMTP System Procedures [page 1515]

xp_startmail System Procedure [page 1880]
xp_stopmail System Procedure [page 1885]
xp_sendmail System Procedure [page 1874]
xp_startsmtp System Procedure [page 1881]
CALL Statement [page 795]

1.6.8.172 xp_write_file System Procedure

Writes data to a file from a SQL statement.

Syntax

```
xp_write_file(
filename
, file_contents
)
```

Parameters

filename

Use this LONG VARCHAR parameter to specify the file name.

file_contents

Use this LONG BINARY parameter to specify the contents to write to the file.

Returns

This function returns an INTEGER status code.

Remarks

The function writes `file_contents` to the file `filename`. It returns 0 if successful, and non-zero if it fails.

The `filename` value can be prefixed by either an absolute or a relative path. If `filename` is prefixed by a relative path, then the file name is relative to the current working directory of the database server. If the file already exists, its contents are overwritten.

This function can be useful for unloading long binary data into files.

You can also use the CSCONVERT function to address character set conversion requirements you have when using the xp_write_file system procedure.

If disk sandboxing is enabled, the file referenced in `filename` must in an accessible location.

Privileges

You must have EXECUTE privilege on the system procedure, as well as the WRITE FILE system privilege.

Example

This example uses xp_write_file to create a file `accountnum.txt` containing the data 123456:

```
CALL dbo.xp_write_file( 'accountnum.txt', '123456' );
```

This example queries the Contacts table of the sample database, and then creates a text file for each contact living in New Jersey. Each text file is named using a concatenation of the contact's first name (GivenName), last name (Surname), and then the string .txt (for example, `Reeves_Scott.txt`), and contains the contact's street address (Street), city (City), and state (State), on separate lines.

```
SELECT dbo.xp_write_file(
Surname || '_' ||  GivenName || '.txt',
Street || '\n' || City || '\n' || State )
FROM Contacts WHERE State = 'NJ';
```

This example uses xp_write_file to create an image file (JPG) for every product in the Products table. Each value of the ID column becomes a file name for a file with the contents of the corresponding value of the Photo column:

```
SELECT dbo.xp_write_file( ID || '.jpg', Photo ) FROM Products;
```

In the example above, ID is a row with a UNIQUE constraint. This is important to ensure that a file isn't overwritten with the contents of subsequent row. Also, you must specify the file extension applicable to the data stored in the column. In this case, the Products.Photo column stores image data (JPEGs).

Related Information

Insertion of Documents and Images
Disk Sandboxing
CSCONVERT Function [String] [page 309]
xp_read_file System Procedure [page 1871]
Directory and File System Procedures [page 1516]

1.7 Views

There are several types of views supported by the software.

In this section:

System Views [page 1889]
The catalog contains system tables that link together by keys and indexes. The system tables are hidden. However, there is a system view for each table. A system view can also include columns from more than one system table to satisfy a commonly needed join.

Consolidated Views [page 1973]
Consolidated views provide data in a form more frequently required by users.

Compatibility Views [page 1998]
Compatibility views are views that are provided for compatibility with versions of the software that are 10 and earlier. Where possible use system and consolidated views instead, as support may diminish for some compatibility views in future releases.

Views for Transact-SQL Compatibility [page 2011]
The Adaptive Server Enterprise and SQL Anywhere system catalogs are different.

1.7.1 System Views

The catalog contains system tables that link together by keys and indexes. The system tables are hidden. However, there is a system view for each table. A system view can also include columns from more than one system table to satisfy a commonly needed join.

To ensure compatibility with future versions of the catalog, applications must make use of system views and not the underlying system tables, which may change.

In SQL Anywhere, the system owner (user SYS) owns the system views.

In this section:

Viewing Detailed System Information for Views and Definitions [page 1896]
Access information about system views, including their definitions, from *SQL Central*.

GTSYSPERFCACHEPLAN System View [page 1896]
Each row in the GTSYSPERFCACHEPLAN system view contains a graphical plan string for an execution plan of the specified statement.

GTSYSPERFCACHESTMT System View [page 1897]
Each row in the GTSYSPERFCACHESTMT system view represents SQL text for a statement with the constants removed.

SYSARTICLE System View [page 1898]
Each row of the SYSARTICLE system view describes an article in a publication. The underlying system table for this view is ISYSARTICLE.

SYSARTICLECOL System View [page 1899]
Each row of the SYSARTICLECOL system view identifies a column in an article. The underlying system table for this view is ISYSARTICLECOL.

The SYSEVENTTYPE system view defines the system event types that can be referenced by CREATE EVENT.

SYSEXTERNENV System View [page 1908]
Each row in the SYSEXTERNENV system view describes the information needed to identify and launch each of the external environments. The underlying system table for this view is ISYSEXTERNENV.

SYSEXTERNENVOBJECT System View [page 1910]
Each row in the SYSEXTERNENVOBJECT system view describes an installed external object. The underlying system table for this view is ISYSEXTERNENVOBJECT.

SYSEXTERNLOGIN System View [page 1911]
Each row in the SYSEXTERNLOGIN system view describes an external login for remote data access. The underlying system table for this view is ISYSEXTERNLOGIN.

SYSEXTERNLOGINPASSWORD System View [page 1911]
Each row in the SYSEXTERNLOGINPASSWORD system view gives the object ID and password hash for an external login. The underlying system table for this view is ISYSEXTERNLOGIN.

SYSFKEY System View [page 1912]
Each row in the SYSFKEY system view describes a foreign key constraint in the system. The underlying system table for this view is ISYSFKEY.

SYSHISTORY System View [page 1914]
Each row in the SYSHISTORY system view records a system operation on the database, such as a database start, a database calibration, and so on. The underlying system table for this view is ISYSHISTORY.

SYSIDX System View [page 1916]
Each row in the SYSIDX system view defines a logical index in the database. The underlying system table for this view is ISYSIDX.

SYSIDXCOL System View [page 1918]
Each row in the SYSIDXCOL system view describes one column of an index described in the SYSIDX system view. The underlying system table for this view is ISYSIDXCOL.

SYSJAR System View [page 1918]
Each row in the SYSJAR system view defines a JAR file stored in the database. The underlying system table for this view is ISYSJAR.

SYSJARCOMPONENT System View [page 1919]
Each row in the SYSJARCOMPONENT system view defines a JAR file component, which includes class files, manifest files, and any other JAR resource. The underlying system table for this view is ISYSJARCOMPONENT.

SYSJAVACLASS System View [page 1920]
Each row in the SYSJAVACLASS system view describes one Java class stored in the database. The underlying system table for this view is ISYSJAVACLASS.

SYSLDAPSERVER System View [page 1920]
The SYSLDAPSERVER system view contains one row for each LDAP server configuration object configured in the database. The underlying system table for this view is ISYSLDAPSERVER.

SYSLDAPSERVERPASSWORD System View [page 1922]
Each row in the SYSLDAPSERVERPASSWORD system view gives the object ID and password hash for an LDAP access account. The underlying system table for this view is ISYSLDAPSERVER.

SYSLOGINMAP System View [page 1922]

The SYSLOGINMAP system view contains one row for each user that can connect to the database using either an integrated login, or Kerberos login. For that reason, access to this view is restricted. The underlying system table for this view is ISYSLOGINMAP.

SYSLOGINPOLICY System View [page 1923]
The underlying system table for this view is ISYSLOGINPOLICY.

SYSLOGINPOLICYOPTION System View [page 1923]
The underlying system table for this view is ISYSLOGINPOLICYOPTION.

SYSMIRROROPTION System View [page 1923]
The underlying system table for this view is ISYSMIRROROPTION.

SYSMIRRORSERVER System View [page 1924]
The underlying system table for this view is ISYSMIRRORSERVER.

SYSMIRRORSERVEROPTION System View [page 1925]
The underlying system table for this view is ISYSMIRRORSERVEROPTION.

SYSMUTEXSEMAPHORE System View [page 1925]
Each row in the SYSMUTEXSEMAPHORE system view provides information about a user-defined mutex or semaphore in the database. The underlying system table for this view is ISYSMUTEXSEMAPHORE.

SYSMVOPTION System View [page 1926]
Each row in the SYSMVOPTION system view describes the setting of one option value for a materialized view or text index at the time of its creation. The name of the option can be found in the SYSMVOPTIONNAME system view. The underlying system table for this view is ISYSMVOPTION.

SYSMVOPTIONNAME System View [page 1927]
Each row in the SYSMVOPTION system view gives the name option value for a materialized view or text index at the time of its creation. The value for the option can be found in the SYSMVOPTION system view. The underlying system table for this view is ISYSMVOPTIONNAME.

SYSOBJECT System View [page 1927]
Each row in the SYSOBJECT system view describes a database object. The underlying system table for this view is ISYSOBJECT.

SYSODATAPRODUCER System View [page 1929]
Each row of the SYSODATAPRODUCER system view describes an OData Producer. The underlying system table for this view is ISYSODATAPRODUCER.

SYSOPTION System View [page 1929]
The SYSOPTION system view contains the options one row for each option setting stored in the database.

SYSOPTSTAT System View [page 1930]
The SYSOPTSTAT system view stores the cost model calibration information as computed by the ALTER DATABASE CALIBRATE statement. The contents of this view are for internal use only and are best accessed via the sa_get_dtt system procedure. The underlying system table for this view is ISYSOPTSTAT.

SYSPHYSIDX System View [page 1930]
Each row in the SYSPHYSIDX system view defines a physical index in the database. The underlying system table for this view is ISYSPHYSIDX.

SYSPROCEDURE System View [page 1931]

Each row in the SYSPROCEDURE system view describes one procedure in the database. The underlying system table for this view is ISYSPROCEDURE.

SYSPROCPARM System View [page 1932]
Each row in the SYSPROCPARM system view describes one parameter, result set column, or return value of a procedure or function in the database. The underlying system table for this view is ISYSPROCPARM.

SYSPROCPERM System View [page 1934]
Each row of the SYSPROCPERM system view describes a user who has been granted EXECUTE privilege on a procedure. The underlying system table for this view is ISYSPROCPERM.

SYSPROXYTAB System View [page 1934]
Each row of the SYSPROXYTAB system view describes the remote parameters of one proxy table. The underlying system table for this view is ISYSPROXYTAB.

SYSPUBLICATION System View [page 1935]
Each row in the SYSPUBLICATION system view describes a publication. The underlying system table for this view is ISYSPUBLICATION.

SYSREMARK System View [page 1936]
Each row in the SYSREMARK system view describes a remark (or comment) for an object. The underlying system table for this view is ISYSREMARK.

SYSREMOTEOPTION System View [page 1936]
Each row in the SYSREMOTEOPTION system view describes the value of a message link parameter. The underlying system table for this view is ISYSREMOTEOPTION.

SYSREMOTEOPTIONTYPE System View [page 1936]
Each row in the SYSREMOTEOPTIONTYPE system view describes one of the message link parameters. The underlying system table for this view is ISYSREMOTEOPTIONTYPE.

SYSREMOTETYPE System View [page 1937]
The SYSREMOTETYPE system view contains information about remote tables. The underlying system table for this view is ISYSREMOTETYPE.

SYSREMOTEUSER System View [page 1937]
Each row in the SYSREMOTEUSER system view describes a user ID with the REMOTE system privilege (a subscriber), together with the status of messages that were sent to and from that user. The underlying system table for this view is ISYSREMOTEUSER.

SYSROLEGRANT System View [page 1939]
The SYSROLEGRANT system view stores information about role membership and type of membership. The underlying system table for this view is ISYSROLEGRANT.

SYSROLEGRANTEXT System View [page 1940]
When you grant the SET USER and the CHANGE PASSWORD system privileges, you can specify a list of users or roles that the grantee can grant them to.

SYSSCHEDULE System View [page 1941]
Each row in the SYSSCHEDULE system view describes a time at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT. The underlying system table for this view is ISYSSCHEDULE.

SYSSEQUENCE System View [page 1942]
The SYSSEQUENCE system view contains one row for each user-defined sequence. The underlying system table for this view is ISYSSEQUENCE.

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

SYSTEXTIDX System View [page 1960]
Each row in the SYSTEXTIDX system view describes one text index. The underlying system table for this view is ISYSTEXTIDX.

SYSTEXTIDXTAB System View [page 1960]
Each row in the SYSTEXTIDXTAB system view describes a generated table that is part of a text index. The underlying system table for this view is ISYSTEXTIDXTAB.

SYSTIMEZONE System View [page 1961]
Each row in the SYSTIMEZONE system view describes one time zone. The underlying system table for this view is ISYSTIMEZONE.

SYSTRIGGER System View [page 1962]
Each row in the SYSTRIGGER system view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE). The underlying system table for this view is ISYSTRIGGER.

SYSTYPEMAP System View [page 1964]
The SYSTYPEMAP system view contains the compatibility mapping values for entries in the SYSSQLSERVERTYPE system view. The underlying system table for this view is ISYSTYPEMAP.

SYSUNITOFMEASURE System View [page 1965]
Each row of the SYSUNITOFMEASURE system view describes a unit of measure defined in the database. The underlying table for the SYSUNITOFMEASURE system view is the ISYSUNITOFMEASURE system table.

SYSUSER System View [page 1965]
Each row in the SYSUSER system view describes a user in the system.

SYSUSERPASSWORD System View [page 1967]
Each row in the SYSUSERPASSWORD system view gives the object ID and password hash for a login account. The underlying system table for this view is ISYSUSER.

SYSUSERMESSAGE System View [page 1968]
Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.

SYSUSERTYPE System View [page 1968]
Each row in the SYSUSERTYPE system view holds a description of a user-defined data type. The underlying system table for this view is ISYSUSERTYPE.

SYSDATABASEVARIABLE System View [page 1969]
Each row in the SYSDATABASEVARIABLE system view describes one database-scope variable in the database. The underlying system table for this view is ISYSDATABASEVARIABLE.

SYSVIEW System View [page 1970]
Each row in the SYSVIEW system view describes a view in the database. Additional information about views can also be found in the SYSTAB system view. The underlying system table for this view is ISYSVIEW.

SYSWEBSERVICE System View [page 1972]
Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

1.7.1.1 Viewing Detailed System Information for Views and Definitions

Access information about system views, including their definitions, from *SQL Central*.

Context

System views are updated when a checkpoint occurs.

Procedure

1. Use the *SQL Anywhere 17* plug-in to connect to the database.
2. Right-click the database, and then click *Configure Owner Filter*.
3. Click *SYS*, and then click *OK*.
4. In the left pane, double-click *Views*.
5. In the left pane click a view owned by SYS (this is indicated by the phrase SYS in parenthesis after the name of the view), and in the right pane click the *SQL* tab.
6. Click the *Data* tab.

Results

The view definition appears on the *Data* tab.

1.7.1.2 GTSYSPERFCACHEPLAN System View

Each row in the GTSYSPERFCACHEPLAN system view contains a graphical plan string for an execution plan of the specified statement.

> **i** Note
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Column name	Data type	Description
stmt_hash	UNSIGNED BIGINT	Unique identifier assigned to each query.
plan_hash	UNSIGNED BIGINT	ID assigned to each plan for a given statement.
plan_text	XML NOT NULL	XML representation of the query execution plan.
update	TIMESTAMP	The time the statement was executed. This value is Coordinated Universal Time (UTC) time.

Remarks

A statement can have multiple execution plans represented in the system view. If statement performance summary data is not collected, then no statement plans are reported.

Plans are not recorded for statements with short (0.005 seconds or less) execution times.

Privileges

You must have the MONITOR system privilege to access this view.

Related Information

Tip: Identify the Cause of Slow Statements
Separately Licensed Components
GTSYSPERFCACHESTMT System View [page 1897]

1.7.1.3 GTSYSPERFCACHESTMT System View

Each row in the GTSYSPERFCACHESTMT system view represents SQL text for a statement with the constants removed.

> **i** Note
>
> Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were

previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See SQL Anywhere Monitor Non-GUI User Guide.

Column name	Data type	Description
stmt_hash	UNSIGNED BIGINT	A unique identifier assigned to each statement.
stmt_text	LONG VARCHAR	SQL representation of the statement.
update	TIMESTAMP	The time the statement was executed. This value is Coordinated Universal Time (UTC) time.

Remarks

The statement performance summary feature uses the SQL statement stored in this view.

The SQL for short running statements (0.005 seconds or less) is not recorded.

Privileges

You must have the MONITOR system privilege to access this view.

Related Information

Tip: Identify the Cause of Slow Statements
GTSYSPERFCACHEPLAN System View [page 1896]
Separately Licensed Components

1.7.1.4 SYSARTICLE System View

Each row of the SYSARTICLE system view describes an article in a publication. The underlying system table for this view is ISYSARTICLE.

Column name	Data type	Description
publication_id	UNSIGNED INT	The publication of which the article is a part.

Column name	Data type	Description
table_id	UNSIGNED INT	Each article consists of columns and rows from a single table. This column contains the table ID for this table.
where_expr	LONG VARCHAR	For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.
subscribe_by_expr	LONG VARCHAR	For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.
query	CHAR(1)	Indicates information about the article type to the database server.
alias	VARCHAR(256)	The alias for the article.
schema_change_active	BIT	1 if the table and publication are part of a synchronization schema change.

1.7.1.5 SYSARTICLECOL System View

Each row of the SYSARTICLECOL system view identifies a column in an article. The underlying system table for this view is ISYSARTICLECOL.

Column name	Data type	Description
publication_id	UNSIGNED INT	A unique identifier for the publication of which the column is a part.
table_id	UNSIGNED INT	The table to which the column belongs.
column_id	UNSIGNED INT	The column identifier, from the SYSTABCOL system view.

1.7.1.6 SYSCAPABILITY System View

Each row of the SYSCAPABILITY system view specifies the status of a capability on a remote database server. The underlying system table for this view is ISYSCAPABILITY.

Column name	Data type	Description
capid	INTEGER	The ID of the capability, as listed in the SYSCAPABILITYNAME system view.
srvid	UNSIGNED INT	The server to which the capability applies, as listed in the SYSSERVER system view.
capvalue	CHAR(128)	The value of the capability.

Related Information

SYSCAPABILITYNAME System View [page 1900]

1.7.1.7 SYSCAPABILITYNAME System View

Each row in the SYSCAPABILITYNAME system view provides a name for each capability ID in the SYSCAPABILITY system view.

Column name	Data type	Description
capid	INTEGER	A number uniquely identifying the capability.
capname	VARCHAR(32000)	The name of the capability.

Remarks

The SYSCAPABILITYNAME system view is defined using a combination of sa_rowgenerator and the following server properties:

- RemoteCapability
- MaxRemoteCapability

Related Information

List of Database Server Properties
SYSCAPABILITY System View [page 1899]

1.7.1.8 SYSCERTIFICATE System View

Each row of the SYSCERTIFICATE system view stores a certificate in text PEM-format. The underlying system table for this view is ISYSCERTIFICATE.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The ID of the certificate.
cert_name	CHAR(128)	The certificate name.
contents	LONG BINARY	The certificate contents in a compressed form.

Column name	Data type	Description
update_time	TIMESTAMP	The local date and time of the last create or replace.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC date and time of the last create or replace.

1.7.1.9 SYSCHECK System View

Each row in the SYSCHECK system view provides the definition for a named check constraint in a table. The underlying system table for this view is ISYSCHECK.

Column name	Data type	Description
check_id	UNSIGNED INT	A number that uniquely identifies the constraint in the database.
check_defn	LONG VARCHAR	The CHECK expression.

1.7.1.10 SYSCOLPERM System View

The GRANT statement can give UPDATE, SELECT, or REFERENCES privileges to individual columns in a table. Each column with UPDATE, SELECT, or REFERENCES privileges is recorded in one row of the SYSCOLPERM system view. The underlying system table for this view is ISYSCOLPERM.

Column name	Data type	Description
table_id	UNSIGNED INT	The table number for the table containing the column.
grantee	UNSIGNED INT	The ID of the user that has been given the privilege on the column. If the user ID is the PUBLIC role, then all users have the privilege on the column.
grantor	UNSIGNED INT	The ID of the user that granted the privilege.
column_id	UNSIGNED INT	This column number, together with the table_id, identifies the column for which privilege has been granted.
privilege_type	SMALLINT	The number in this column indicates the kind of column privilege (16=REFERENCES, 1=SELECT, or 8=UPDATE).
is_grantable	CHAR(1)	Indicates if the privilege on the column was granted WITH GRANT OPTION.

1.7.1.11 SYSCOLSTAT System View

The SYSCOLSTAT system view contains the column statistics, including histograms, that are used by the optimizer. The contents of this view are best retrieved using the sa_get_histogram stored procedure or the Histogram utility. The underlying system table for this view is ISYSCOLSTAT.

Column name	Data type	Description
table_id	UNSIGNED INT	A number that uniquely identifies the table or materialized view to which the column belongs.
column_id	UNSIGNED INT	A number that, together with table_id, uniquely identifies the column.
format_id	SMALLINT	For system use only.
update_time	TIMESTAMP	The local time of the last update of the column statistics.
density	FLOAT	An estimate of the average selectivity of a single value for the column, not counting the large single value selectivities stored in the row.
max_steps	SMALLINT	For system use only.
actual_steps	SMALLINT	For system use only.
step_values	LONG BINARY	For system use only.
frequencies	LONG BINARY	For system use only.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time of the last update of the column statistics.

> **i Note**
>
> For databases created using SQL Anywhere 16 or later, the underlying system table for this view is always encrypted to protect the data from unauthorized access.

Related Information

sa_get_histogram System Procedure [page 1601]
Histogram Utility (dbhist)

1.7.1.12 SYS.SYSCONSTRAINT System View [Relational Data Lake]

Each row in the SYS.SYSCONSTRAINT system view describes a named constraint in the database. The underlying system table for this view is ISYSCONSTRAINT.

Column name	Data type	Description
constraint_id	UNSIGNED INT	The unique ID for the constraint.
constraint_type	CHAR(1)	The type of constraint: **C** column check constraint **T** table constraint **P** primary key **F** foreign key **U** unique constraint
ref_object_id	UNSIGNED BIGINT	The object ID of the column, table, or index to which the constraint applies.
table_object_id	UNSIGNED BIGINT	The object ID of the table to which the constraint applies.
constraint_name	CHAR(128)	The name of the constraint.

1.7.1.13 SYSDBFILE System View

Each row in the SYSDBFILE system view describes a dbspace file. The underlying system table for this view is ISYSDBFILE.

Column name	Data type	Description
dbfile_id	SMALLINT	For internal use only.
dbspace_id	SMALLINT	Each dbspace file in a database is assigned a unique number. The system dbspace contains all system objects and has a dbspace_id of 0.
dbfile_name	CHAR(128)	A unique name for the dbspace. It is used in the CREATE TABLE command.
file_name	LONG VARCHAR	The file name for the dbspace.
lob_map	LONG VARBIT	For internal use only.

Column name	Data type	Description
server_id	UNSIGNED INT	For internal use only.

1.7.1.14 SYSDBSPACE System View

Each row in the SYSDBSPACE system view describes a dbspace file. The underlying system table for this view is ISYSDBSPACE.

Column name	Data type	Description
dbspace_id	SMALLINT	Unique number identifying the dbspace. The system dbspace contains all system objects and has a dbspace_id of 0.
object_id	UNSIGNED BIGINT	The object ID of the dbspace.
dbspace_name	CHAR(128)	A unique name for the dbspace. It is used in the CREATE TABLE command.
store_type	TINYINT	For internal use only.
saved_cache_pages	LONG VARBIT	The pages of the dbspace that were present in the cache when the ALTER DATABASE SAVE CACHE statement was most recently executed. The value is NULL if the statement has never been executed.

1.7.1.15 SYSDBSPACEPERM System View

Each row in the SYSDBSPACEPERM system view describes a privilege on a dbspace file. The underlying system table for this view is ISYSDBSPACEPERM.

Column name	Data type	Description
dbspace_id	SMALLINT	Unique number identifying the dbspace. The system dbspace contains all system objects and has a dbspace_id of 0.
grantee	UNSIGNED INT	The user ID of the user getting the privilege.
privilege_type	SMALLINT	The privilege that is granted to the grantee. For example, CREATE gives the grantee privilege to create objects on the dbspace.

Related Information

User Security (Roles and Privileges)
GRANT Statement [page 1193]

1.7.1.16 SYSDEPENDENCY System View

Each row in the SYSDEPENDENCY system view describes a dependency between two database objects. The underlying system table for this view is ISYSDEPENDENCY.

A dependency exists between two database objects when one object references another object in its definition. For example, if the query specification for a view references a table, the view is dependent on the table. The database server tracks dependencies of views on tables, views, materialized views, and columns.

Column name	Data type	Description
ref_object_id	UNSIGNED BIGINT	The object ID of the referenced object.
dep_object_id	UNSIGNED BIGINT	The ID of the referencing object.

Related Information

View Dependencies
sa_dependent_views System Procedure [page 1571]

1.7.1.17 SYSDOMAIN System View

The SYSDOMAIN system view records information about built-in data types (also called domains). The contents of this view does not change during normal operation. The underlying system table for this view is ISYSDOMAIN.

Column name	Data type	Description
domain_id	SMALLINT	The unique number assigned to each data type. These numbers cannot be changed.
domain_name	CHAR(128)	The name of the data type normally found in the CREATE TABLE command, such as CHAR or INTEGER.
type_id	SMALLINT	The ODBC data type. This value corresponds to the value for data_type in the Transact-SQL compatibility dbo.SYSTYPES table.

Column name	Data type	Description
"precision"	SMALLINT	The number of significant digits that can be stored using this data type. The column value is NULL for non-numeric data types.

1.7.1.18 SYSEVENT System View

Each row in the SYSEVENT system view describes an event created with CREATE EVENT. The underlying system table for this view is ISYSEVENT.

Column name	Data type	Description
event_id	UNSIGNED INT	The unique number assigned to each event.
object_id	UNSIGNED BIGINT	The internal ID for the event, uniquely identifying it in the database.
creator	UNSIGNED INT	The user number of the owner of the event. The name of the user can be found by looking in the SYSUSER system view.
event_name	VARCHAR(128)	The name of the event.
enabled	CHAR(1)	Indicates whether the event is allowed to fire.

Column name	Data type	Description
location	CHAR(1)	The location where the event is to fire: • Y = AT ALL clause and FOR PRIMARY clause specified • E = AT CONSOLIDATED clause and FOR PRIMARY clause specified • T = AT REMOTE clause and FOR PRIMARY clause specified • P = (AT clause not specified) FOR PRIMARY clause specified • B = AT ALL clause and FOR ALL clause specified • D = AT CONSOLIDATED clause and FOR ALL clause specified • S = AT REMOTE clause and FOR ALL clause specified • M = (AT clause not specified) FOR ALL clause specified • C = AT CONSOLIDATED (FOR clause not specified) • R = AT REMOTE (FOR clause not specified) • A = AT ALL clause (FOR clause not specified)
event_type_id	UNSIGNED INT	For system events, the event type as listed in the SYSEVENTTYPE system view.
action	LONG VARCHAR	The event handler definition. An obfuscated value indicates a hidden event.
external_action	LONG VARCHAR	For system use only.
condition	LONG VARCHAR	The condition used to control firing of the event handler.
remarks	LONG VARCHAR	Remarks for the event; this column comes from ISYSREMARK.
source	LONG VARCHAR	The original source for the event; this column comes from ISYSSOURCE.

Related Information

SYSEVENTTYPE System View [page 1908]

1.7.1.19 SYSEVENTTYPE System View

The SYSEVENTTYPE system view defines the system event types that can be referenced by CREATE EVENT.

Column name	Data type	Description
event_type_id	INT	The unique number assigned to each event type.
name	VARCHAR(32000)	The name of the system event type.
description	VARCHAR(32000)	A description of the system event type.

Remarks

The SYSEVENTTYPE system view is defined using a combination of sa_rowgenerator and the following server properties:

- EventTypeName
- EventTypeDesc
- MaxEventType

Related Information

List of Database Server Properties
SYSEVENT System View [page 1906]

1.7.1.20 SYSEXTERNENV System View

Each row in the SYSEXTERNENV system view describes the information needed to identify and launch each of the external environments. The underlying system table for this view is ISYSEXTERNENV.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	A unique identifier for the external environment.
name	CHAR(128)	This column identifies the name of the external environment or language. It is one of *java*, *perl*, *php*, and so on.

Column name	Data type	Description
scope	CHAR(1)	This column is either C for CONNECTION or D for DATABASE respectively. The scope column identifies if the external environment is launched as one-per-connection or one-per-database. For one-per-connection external environments (like PERL, PHP, JS, C_ESQL32, C_ESQL64, C_ODBC32, and C_ODBC64), there is one instance of the external environment for each connection using the external environment. For a one-per-connection, the external environment terminates when the connection terminates. For one-per-database external environments (such as JAVA), there is one instance of the external environment for each database using the external environment. The one-per-database external environment terminates when the database is stopped.
support_result_sets	CHAR(1)	This column identifies those external environments that can return result sets. All external environments can return result sets except PERL, PHP, and JS.
location	LONG VARCHAR	This column identifies the location on the database server computer where the executable/binary for the external environment can be found. It includes the executable/binary name. This path can either be fully qualified or relative. If the path is relative, then the executable/binary must be in a location where the database server can find it.
options	LONG VARCHAR	This column identifies any options required on the command line to launch the executable associated with the external environment. Do not modify this column.

Column name	Data type	Description
user_id	UNSIGNED INT	When the external environment is initially launched, it must make a connection back to the database to set things up for the external environment's usage. By default, this connection is made using the DBA user ID, but if the database administrator prefers to have the external environment use a different user ID with MANAGE ANY EXTERNAL OBJECT system privilege, then the user_id column would indicate that different user ID instead. Typically, this column is NULL and the database server, by default, uses the DBA user ID.

Related Information

External Environment Support

1.7.1.21 SYSEXTERNENVOBJECT System View

Each row in the SYSEXTERNENVOBJECT system view describes an installed external object. The underlying system table for this view is ISYSEXTERNENVOBJECT.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	A unique identifier for the external object.
extenv_id	UNSIGNED BIGINT	The unique identifier for the external environment (SYSEXTERNENV.object_id).
owner	UNSIGNED INT	This column identifies the creator/owner of the external object.
name	LONG VARCHAR	This column identifies the name of the external object as specified in the INSTALL EXTERNAL OBJECT statement.
contents	LONG BINARY	The contents of the external object.
update_time	TIMESTAMP	This column identifies the last local time the object was modified (or installed).
update_time_utc	TIMESTAMP WITH TIME ZONE	This column identifies the last UTC time the object was modified (or installed).

Related Information

External Environment Support

1.7.1.22 SYSEXTERNLOGIN System View

Each row in the SYSEXTERNLOGIN system view describes an external login for remote data access. The underlying system table for this view is ISYSEXTERNLOGIN.

Column name	Data type	Description
user_id	UNSIGNED INT	The user ID on the local database.
srvid	UNSIGNED INT	The remote server, as listed in the SYSSERVER system view.
remote_login	VARCHAR(128)	The login name for the user, for the remote server.
remote_password	CHAR(3)	Whether a password is stored. Three asterisks (***) indicate that a password is stored. NULL indicates that no password is stored. You can obtain the password hash by querying the SYSEXTERNLOGINPASSWORD system view.

Previous versions of the catalog contained a SYSEXTERNLOGINS system table. That table has been renamed to be ISYSEXTERNLOGIN (without an 'S'), and is the underlying table for this view.

Related Information

SYSEXTERNLOGINPASSWORD System View [page 1911]

1.7.1.23 SYSEXTERNLOGINPASSWORD System View

Each row in the SYSEXTERNLOGINPASSWORD system view gives the object ID and password hash for an external login. The underlying system table for this view is ISYSEXTERNLOGIN.

You must have the SELECT ANY TABLE and ACCESS USER PASSWORD system privileges to access this view.

Column	Data type	Description
user_id	UNSIGNED INT	The user ID on the local database.
srvid	UNSIGNED INT	The remote server, as listed in the SYSSERVER system view.

Column	Data type	Description
remote_password	VARBINARY(128)	The password hash for the external login.

Related Information

SYSEXTERNLOGIN System View [page 1911]

1.7.1.24 SYSFKEY System View

Each row in the SYSFKEY system view describes a foreign key constraint in the system. The underlying system table for this view is ISYSFKEY.

Column name	Data type	Description
foreign_table_id	UNSIGNED INT	The table number of the foreign table.
foreign_index_id	UNSIGNED INT	The index number for the foreign key.
primary_table_id	UNSIGNED INT	The table number of the primary table.
primary_index_id	UNSIGNED INT	The index number of the primary key.
match_type	TINYINT	The matching type for the constraint. Matching types include: **0** Use the default matching **1** SIMPLE **2** FULL **129** SIMPLE UNIQUE **130** FULL UNIQUE For more information about match types, see the MATCH clause of the CREATE TABLE statement.
check_on_commit	CHAR(1)	Indicates whether INSERT and UPDATE statements should wait until the COMMIT to check if foreign keys are still valid. Values are 'Y' for wait or 'N' for do not wait.

Column name	Data type	Description
nulls	CHAR(1)	Indicates whether the columns in the foreign key are allowed to contain the NULL value. This setting is independent of the nulls setting in the columns contained in the foreign key. Values are 'Y' for allowed or 'N' for not allowed.

Related Information

CREATE TABLE Statement [page 1002]

1.7.1.25 SYSHISTORY System View

Each row in the SYSHISTORY system view records a system operation on the database, such as a database start, a database calibration, and so on. The underlying system table for this view is ISYSHISTORY.

Column name	Data type	Description
operation	CHAR(128)	The type of operation performed on the database file. The operation must be one of the following values: **INIT** Information about when the database was created. **UPGRADE** Information about when the database was upgraded. **START** Information about when the database was started using a specific version of the database server on a particular operating system. **LAST_START** Information about the most recent time the database server was started. A LAST_START operation is converted to a START operation when the database is started with a different version of the database server and/or on a different operating system than those values currently stored in the LAST_START row. **DTT** Information about the *second to last* Disk Transfer Time (DTT) calibration operation performed on the dbspace. That is, information about the second to last execution of either an ALTER DATABASE CALIBRATE or ALTER DATABASE RESTORE DEFAULT CALIBRATION statement. **LAST_DTT** Information about the *most recent* DTT calibration operation per-

Column name	Data type	Description
		formed on the dbspace. That is, information about the most recent execution of either an ALTER DATABASE CALIBRATE or ALTER DATABASE RESTORE DEFAULT CALIBRATION statement. **LAST_BACKUP** Information about the last backup, including date and time of the backup, the backup type, the files that were backed up, and the version of database server that performed the backup.
object_id	UNSIGNED INT	For any operation other than DTT and LAST_DTT, the value in this column will be 0. For DTT and LAST_DTT operations, this is the dbspace_id of the dbspace as defined in the SYSDBSPACE system view.
sub_operation	CHAR(128)	For any operation other than DTT and LAST_DTT, the value in this column will be a set of empty single quotes ('). For DTT and LAST_DTT operations, this column contains the type of sub-operation performed on the dbspace. Values include: **DTT_SET** The dbspace calibration has been set. **DTT_UNSET** The dbspace calibration has been restored to the default setting.
version	CHAR(128)	The version and build number of the database server used to perform the operation.
platform	CHAR(128)	The operating system on which the operation was carried out.
first_time	TIMESTAMP	The local date and time the database was first started on a particular operating system with a particular version of the software.

Column name	Data type	Description
last_time	TIMESTAMP	The most recent local date and time the database was started on a particular operating system with a particular version of the software.
details	LONG VARCHAR	This column stores information such as command line options used to start the database server or the capability bits enabled for the database. This information is for use by Technical Support.
first_time_utc	TIMESTAMP WITH TIME ZONE	The UTC date and time the database was first started on a particular operating system with a particular version of the software.
last_time_utc	TIMESTAMP WITH TIME ZONE	The most recent UTC date and time the database was started on a particular operating system with a particular version of the software.

Related Information

SYSDBSPACE System View [page 1904]

1.7.1.26 SYSIDX System View

Each row in the SYSIDX system view defines a logical index in the database. The underlying system table for this view is ISYSIDX.

Column name	Data type	Description
table_id	UNSIGNED INT	Uniquely identifies the table to which this index applies.
index_id	UNSIGNED INT	A unique number identifying the index within its table.
object_id	UNSIGNED BIGINT	The internal ID for the index, uniquely identifying it in the database.
phys_index_id	UNSIGNED INT	Identifies the underlying physical index used to implement the logical index. This value is NULL for indexes on temporary tables or remote tables. Otherwise, the value corresponds to the object_id of a physical index in the SYSPHYSIDX system view.

Column name	Data type	Description
dbspace_id	SMALLINT	The ID of the file in which the index is contained. This value corresponds to an entry in the SYSDBSPACE system view.
index_category	TINYINT	The type of index. Values include: **1** Primary key **2** Foreign key **3** Secondary index (includes unique constraints) **4** Text indexes
"unique"	TINYINT	Indicates whether the index is a unique index (1), a unique constraint (2), reserved (3), a non-unique index (4), or a unique index WITH NULLS NOT DISTINCT (5).
index_name	CHAR(128)	The name of the index.
not_enforced	CHAR(1)	For internal use only.
file_id	SMALLINT	DEPRECATED. This column is present in SYSVIEW, but not in the underlying system table ISYSIDX. The contents of this column are the same as dbspace_id, and it is provided for compatibility. Use dbspace_id instead.

Related Information

SYSIDXCOL System View [page 1918]
SYSPHYSIDX System View [page 1930]
SYSDBSPACE System View [page 1904]

1.7.1.27 SYSIDXCOL System View

Each row in the SYSIDXCOL system view describes one column of an index described in the SYSIDX system view. The underlying system table for this view is ISYSIDXCOL.

Column name	Data type	Description
table_id	UNSIGNED INT	Identifies the table to which the index applies.
index_id	UNSIGNED INT	Identifies the index to which the column applies. Together, table_id and index_id identify one index described in the SYSIDX system view.
sequence	SMALLINT	Each column in an index is assigned a unique number starting at 0. The order of these numbers determines the relative significance of the columns in the index. The most important column has sequence number 0.
column_id	UNSIGNED INT	Identifies which column of the table is indexed. Together, table_id and column_id identify one column described in the SYSCOLUMN system view.
"order"	CHAR(1)	Indicates whether the column in the index is kept in ascending(A) or descending(D) order. This value is NULL for text indexes.
primary_column_id	UNSIGNED INT	The ID of the primary key column that corresponds to this foreign key column. The value is NULL for non foreign key columns.

Related Information

SYSIDX System View [page 1916]

1.7.1.28 SYSJAR System View

Each row in the SYSJAR system view defines a JAR file stored in the database. The underlying system table for this view is ISYSJAR.

Column name	Data type	Description
jar_id	INTEGER	A unique number identifying the JAR file.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the JAR file, uniquely identifying it in the database.
creator	UNSIGNED INT	The user number of the creator of the JAR file. Can be set by the AS USER clause of the INSTALL JAVA statement.
jar_name	LONG VARCHAR	The name of the JAR file.
jar_file	LONG VARCHAR	This column is no longer used and contains NULL.
update_time	TIMESTAMP	The local time the JAR file was last updated.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time the JAR file was last updated.

Related Information

SYSJARCOMPONENT System View [page 1919]
SYSJAVACLASS System View [page 1920]

1.7.1.29 SYSJARCOMPONENT System View

Each row in the SYSJARCOMPONENT system view defines a JAR file component, which includes class files, manifest files, and any other JAR resource. The underlying system table for this view is ISYSJARCOMPONENT.

Column name	Data type	Description
component_id	INTEGER	The primary key containing the id of the component.
jar_id	INTEGER	If the row describes a JAR, the value is the ID number of the JAR. Otherwise, the value is NULL.
component_name	LONG VARCHAR	The name of the component.
component_type	CHAR(1)	This column is no longer used and contains NULL.
contents	LONG BINARY	The contents of the JAR file component. For a manifest-like component, this is character text. For a class file component, this is byte code.

Related Information

SYSJAR System View [page 1918]
SYSJAVACLASS System View [page 1920]

1.7.1.30 SYSJAVACLASS System View

Each row in the SYSJAVACLASS system view describes one Java class stored in the database. The underlying system table for this view is ISYSJAVACLASS.

Column name	Data type	Description
class_id	INTEGER	The unique number for the Java class. Also the primary key for the table.
object_id	UNSIGNED BIGINT	The internal ID for the Java class, uniquely identifying it in the database.
creator	UNSIGNED INT	The user number of the creator of the class. Can be set by the AS USER clause of the INSTALL JAVA statement.
jar_id	INTEGER	The id of the JAR file from which the class came.
class_name	LONG VARCHAR	The name of the Java class.
public	CHAR(1)	Indicates whether the class is public (Y) or private (N).
component_id	INTEGER	The id of the component in the SYSJAR-COMPONENT system view.
update_time	TIMESTAMP	The local last update time of the class.
update_time_utc	TIMESTAMP WITH TIME ZONE	The UTC last update time of the class.

Related Information

SYSJAR System View [page 1918]
SYSJARCOMPONENT System View [page 1919]

1.7.1.31 SYSLDAPSERVER System View

The SYSLDAPSERVER system view contains one row for each LDAP server configuration object configured in the database. The underlying system table for this view is ISYSLDAPSERVER.

An LDAP server configuration object contains the configuration information necessary to connect to an external LDAP server.

Column name	Data type	Description
ldsrv_id	UNSIGNED BIGINT	A unique identifier for the LDAP server. This ID is used by the login policy to refer to this LDAP server. This column is a foreign key to ISYSOBJECT.
ldsrv_name	CHAR(128)	The name of the LDAP server.
ldsrv_state	CHAR(9)	The state of the LDAP server. Valid values: • RESET • READY • ACTIVE • FAILED • SUSPENDED
ldsrv_start_tls	TINYINT	The valid values: 1 (ON) or 0 (OFF). If ON then LDAP over TLS is used to connect to the LDAP server. This protocol provides encrypted communication for connections and searches with the LDAP server.
ldsrv_num_retries	TINYINT	The number of attempts to authenticate with the LDAP server before returning failure or failover (if specified). Valid range: 1-60.
ldsrv_timeout	UNSIGNED INT	The timeout value for connections or searches, in milliseconds. Valid range: 1-3600000 (1 hour).
ldsrv_last_state_change	TIMESTAMP	The time when the last state change occurred. Regardless of the server's local time zone, the value is stored in Coordinated Universal Time (UTC).
ldsrv_search_url	CHAR(1024)	The LDAP URL defining the search to find the Distinguished Name (DN) for a user based on the user ID.
ldsrv_auth_url	CHAR(1024)	The LDAP search string to find the DN for a user given their user ID.
ldsrv_access_dn	CHAR(1024)	The DN used to access the LDAP server for searches to obtain DNs for other user IDs.
ldsrv_access_dn_pwd	CHAR(3)	Whether a password is stored. Three asterisks (***) indicate that a password is stored. NULL indicates that no password is stored. You can obtain password hash information by querying the SYSLDAPSERVERPASSWORD system view.

Related Information

SYSLDAPSERVERPASSWORD System View [page 1922]
trusted_certificates_file Option

1.7.1.32 SYSLDAPSERVERPASSWORD System View

Each row in the SYSLDAPSERVERPASSWORD system view gives the object ID and password hash for an LDAP access account. The underlying system table for this view is ISYSLDAPSERVER.

You must have the SELECT ANY TABLE and ACCESS USER PASSWORD system privileges to access this view.

Column	Data type	Description
ldsrv_id	UNSIGNED BIGINT	A unique identifier for the LDAP server. This ID is used by the login policy to refer to this LDAP server.
ldsrv_access_dn_pwd	VARBINARY(1024)	The password hash for the LDAP access account.

Related Information

SYSLDAPSERVER System View [page 1920]

1.7.1.33 SYSLOGINMAP System View

The SYSLOGINMAP system view contains one row for each user that can connect to the database using either an integrated login, or Kerberos login. For that reason, access to this view is restricted. The underlying system table for this view is ISYSLOGINMAP.

Column name	Data type	Description
login_mode	TINYINT	The type of login: 1 for integrated logins, 2 for Kerberos logins.
login_id	VARCHAR(1024)	Either the integrated login user profile name, or the Kerberos principal that maps to database_uid.
object_id	UNSIGNED BIGINT	A unique identifier, one for each mapping between user ID and database user ID.
database_uid	UNSIGNED INT	The database user ID to which the login ID is mapped.

1.7.1.34 SYSLOGINPOLICY System View

The underlying system table for this view is ISYSLOGINPOLICY.

Column name	Data type	Description
login_policy_id	UNSIGNED BIGINT	A unique identifier for the login policy.
login_policy_name	CHAR(128)	The name of the login policy.

Related Information

SYSLOGINPOLICYOPTION System View [page 1923]
SYSUSER System View [page 1965]

1.7.1.35 SYSLOGINPOLICYOPTION System View

The underlying system table for this view is ISYSLOGINPOLICYOPTION.

Column name	Data type	Description
login_policy_id	UNSIGNED BIGINT	A unique identifier for the login policy.
login_option_name	CHAR(128)	The name of the login policy.
login_option_value	LONG VARCHAR	The value of the login policy at the time it was created.

Related Information

SYSLOGINPOLICY System View [page 1923]
SYSUSER System View [page 1965]

1.7.1.36 SYSMIRROROPTION System View

The underlying system table for this view is ISYSMIRROROPTION.

Column name	Data type	Description
option_name	CHAR(128)	The name of the option.

Column name	Data type	Description
option_value	LONG VARCHAR	The value of the option when the mirror was created. Values in this column are hidden from users that do not have the SELECT ANY TABLE system privilege.

Related Information

SYSMIRRORSERVER System View [page 1924]
SYSMIRRORSERVEROPTION System View [page 1925]
SET MIRROR OPTION Statement [page 1381]

1.7.1.37 SYSMIRRORSERVER System View

The underlying system table for this view is ISYSMIRRORSERVER.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	A unique identifier for the mirror server.
server_name	CHAR(128)	The name of the server.
server_type	CHAR(20)	The type of server. The value can be one of PRIMARY, MIRROR, ARBITER, PARTNER, or COPY.
parent	UNSIGNED BIGINT	The parent server. If the value is NULL, then the server is the primary or mirror server in a database mirroring system. If there is a value in this column, it is the ID of the server that is the parent of the current server.
alternate_parent	UNSIGNED BIGINT	The ID of the server that is used as an alternate parent if the current parent becomes unavailable.

Related Information

Copy Node Parent Determination
SYSMIRROROPTION System View [page 1923]
SYSMIRRORSERVEROPTION System View [page 1925]

CREATE MIRROR SERVER Statement [page 901]

1.7.1.38 SYSMIRRORSERVEROPTION System View

The underlying system table for this view is ISYSMIRRORSERVEROPTION.

Column name	Data type	Description
server_id	UNSIGNED BIGINT	A unique identifier for the mirror server.
option_name	CHAR(128)	The name of the option.
option_value	LONG VARCHAR	The value of the option when the mirror was created.

Related Information

SYSMIRROROPTION System View [page 1923]
SYSMIRRORSERVER System View [page 1924]

1.7.1.39 SYSMUTEXSEMAPHORE System View

Each row in the SYSMUTEXSEMAPHORE system view provides information about a user-defined mutex or semaphore in the database. The underlying system table for this view is ISYSMUTEXSEMAPHORE.

You must have the SELECT ANY TABLE privilege to access this view.

Column	Data type	Description
mutex_semaphore_id	UNSIGNED INT	A unique ID for the mutex or semaphore.
object_id	UNSIGNED INT	The object ID of the mutex or semaphore in the ISYSOBJECT system table.
owner	UNSIGNED INT	The owner of the mutex or semaphore.
name	CHAR(128)	The name of the mutex or semaphore.
obj_type	CHAR(9)	The type of object: either MUTEX or SEMAPHORE.

Column	Data type	Description
scope	CHAR(11)	The scope for the mutex or semaphore. For mutexes, CONNECTION indicates a connection-level scope, and TRANSACTION indicates a transaction-level scope. For semaphores, this value is always CONNECTION.
start_with	UNSIGNED INT	The initial counter value for a semaphore. This value is NULL for mutexes.

Related Information

Mutexes and Semaphores

1.7.1.40 SYSMVOPTION System View

Each row in the SYSMVOPTION system view describes the setting of one option value for a materialized view or text index at the time of its creation. The name of the option can be found in the SYSMVOPTIONNAME system view. The underlying system table for this view is ISYSMVOPTION.

Column name	Data type	Description
view_object_id	UNSIGNED BIGINT	The object ID of the materialized view.
option_id	UNSIGNED INT	A unique number identifying the option in the database. To see the option name, see the SYSMVOPTIONNAME system view.
option_value	LONG VARCHAR	The value of the option when the materialized view was created.

Related Information

Viewing Text Index Terms and Settings (SQL Central)
Viewing Materialized View Information in the Catalog
SYSMVOPTIONNAME System View [page 1927]

1.7.1.41 SYSMVOPTIONNAME System View

Each row in the SYSMVOPTION system view gives the name option value for a materialized view or text index at the time of its creation. The value for the option can be found in the SYSMVOPTION system view. The underlying system table for this view is ISYSMVOPTIONNAME.

Column name	Data type	Description
option_id	UNSIGNED INT	A number uniquely identifying the option in the database.
option_name	CHAR(128)	The name of the option.

Related Information

Viewing Text Index Terms and Settings (SQL Central)
Viewing Materialized View Information in the Catalog
SYSMVOPTION System View [page 1926]

1.7.1.42 SYSOBJECT System View

Each row in the SYSOBJECT system view describes a database object. The underlying system table for this view is ISYSOBJECT.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the object, uniquely identifying it in the database.

Column name	Data type	Description
status	TINYINT	The status of the object. Values include: **1 (valid)** The object is available for use by the database server. This status is synonymous with ENABLED. That is, if you ENABLE an object, the status changes to VALID. **2 (invalid)** An attempt to recompile the object after an internal operation has failed, for example, after a schema-altering modification to an object on which it depends. The database server continues to try to recompile the object whenever it is referenced in a statement. **4 (disabled)** The object has been explicitly disabled by the user, for example using an ALTER TABLE...DISABLE VIEW DEPENDENCIES statement.
object_type	TINYINT	Type of object.
creation_time	TIMESTAMP	The local date and time when the object was created.
object_type_str	CHAR (128)	Type of object.
creation_time_utc	TIMESTAMP WITH TIME ZONE	The UTC date and time when the object was created.

Related Information

Statuses for Regular Views
Advanced: Status and Properties for Materialized Views

1.7.1.43 SYSODATAPRODUCER System View

Each row of the SYSODATAPRODUCER system view describes an OData Producer. The underlying system table for this view is ISYSODATAPRODUCER.

Column name	Data type	Description
producer_id	UNSIGNED BIGINT	The ID of the OData Producer.
producer_name	VARCHAR (128)	The name of the OData Producer.
admin_user	VARCHAR (128)	The database user acting as the OData administrator.
auth_user	VARCHAR (128)	The database user whose authentication credentials are used by all users to connect to the database server.
enabled	CHAR(1)	Specifies whether the OData Producer is enabled or disabled.
model	LONG BINARY	Specifies the OData Producer service model (given in OSDL).
service_root	LONG VARCHAR	Specifies the root of the OData service on the OData server.
using_string	LONG VARCHAR	Contains the string given by the USING clause of the CREATE ODATA PRODUCER statement.

1.7.1.44 SYSOPTION System View

The SYSOPTION system view contains the options one row for each option setting stored in the database.

Each user can have their own setting for a given option. In addition, settings for the PUBLIC role define the default settings to be used for users that do not have their own setting. The underlying system table for this view is ISYSOPTION.

Column name	Data type	Description
user_id	UNSIGNED INT	The user number to whom the option setting applies.
"option"	CHAR(128)	The name of the option.
setting	LONG VARCHAR	The current setting for the option.

1.7.1.45 SYSOPTSTAT System View

The SYSOPTSTAT system view stores the cost model calibration information as computed by the ALTER DATABASE CALIBRATE statement. The contents of this view are for internal use only and are best accessed via the sa_get_dtt system procedure. The underlying system table for this view is ISYSOPTSTAT.

Column name	Data type	Description
stat_id	UNSIGNED INT	For system use only.
group_id	UNSIGNED INT	For system use only.
format_id	SMALLINT	For system use only.
data	LONG BINARY	For system use only.

1.7.1.46 SYSPHYSIDX System View

Each row in the SYSPHYSIDX system view defines a physical index in the database. The underlying system table for this view is ISYSPHYSIDX.

Column name	Data type	Description
table_id	UNSIGNED INT	The object ID of the table to which the index corresponds.
phys_index_id	UNSIGNED INT	The unique number of the physical index within its table.
root	INTEGER	Identifies the location of the root page of the physical index in the database file.
key_value_count	UNSIGNED INT	The number of distinct key values in the index.
leaf_page_count	UNSIGNED INT	The number of leaf index pages.
depth	UNSIGNED SMALLINT	The depth (number of levels) of the physical index.
max_key_distance	UNSIGNED INT	For system use only.
seq_transitions	UNSIGNED INT	For system use only.
rand_transitions	UNSIGNED INT	For system use only.
rand_distance	UNSIGNED INT	For system use only.
allocation_bitmap	LONG VARBIT	For system use only.
long_value_bitmap	LONG VARBIT	For system use only.

Related Information

SYSIDX System View [page 1916]

SYSIDXCOL System View [page 1918]

1.7.1.47 SYSPROCEDURE System View

Each row in the SYSPROCEDURE system view describes one procedure in the database. The underlying system table for this view is ISYSPROCEDURE.

Column name	Data type	Description
proc_id	UNSIGNED INT	The internal procedure number for the procedure, uniquely identifying it in the database.
creator	UNSIGNED INT	The owner of the procedure.
object_id	UNSIGNED BIGINT	The internal ID for the procedure, uniquely identifying it in the database.
proc_name	CHAR(128)	The name of the procedure. One creator cannot have two procedures with the same name.
proc_defn	LONG VARCHAR	The definition of the procedure after it has been parsed and unparsed by the database server
remarks	LONG VARCHAR	Remarks about the procedure. This value is stored in the ISYSREMARK system table.
replicate	CHAR(1)	Internal use only.
srvid	UNSIGNED INT	If the procedure is a proxy for a procedure on a remote database server, then this value indicates the remote server.
source	LONG VARCHAR	The preserved source for the procedure. This value is stored in the ISYSSOURCE system table. Content is only stored in this column when the preserve_source_format option is set to On.
avg_num_rows	FLOAT	Information collected for use in query optimization when the procedure appears in the FROM clause.
avg_cost	FLOAT	Information collected for use in query optimization when the procedure appears in the FROM clause.
stats	LONG BINARY	Information collected for use in query optimization when the procedure appears in the FROM clause.
dialect	CHAR(1)	Returns W for Watcom SQL procedures and functions, and T for Transact SQL procedures and functions.

Column name	Data type	Description
is_deterministic	CHAR(1)	Returns NULL for procedures. Returns Y if a function is marked as deterministic, N if it is marked as not deterministic, and U if it is not marked either way.
is_external	CHAR(1)	Returns Y if the function is external and N if it is not.
external_language	VARCHAR(128)	Returns NULL if the procedure is not external and 'native' if the procedure uses the original interface. Otherwise, it contains the language of the procedure.
external_name	VARCHAR(32767)	Returns NULL if the procedure is not external. Otherwise, it contains the external name of the procedure.
sql_security	CHAR(1)	Returns I if the procedure is SQL SECURITY INVOKER and D if the procedure is SQL SECURITY DEFINER.

Related Information

preserve_source_format Option
ERROR_LINE Function [Miscellaneous] [page 355]
ERROR_STACK_TRACE Function [Miscellaneous] [page 362]
sa_error_stack_trace System Procedure [page 1589]
sa_stack_trace System Procedure [page 1725]
STACK_TRACE Function [Miscellaneous] [page 566]

1.7.1.48 SYSPROCPARM System View

Each row in the SYSPROCPARM system view describes one parameter, result set column, or return value of a procedure or function in the database. The underlying system table for this view is ISYSPROCPARM.

Column name	Data type	Description
proc_id	UNSIGNED INT	Uniquely identifies the procedure or function to which the parameter belongs.
parm_id	SMALLINT	Each procedure starts numbering parameters at 1. The order of parameter numbers corresponds to the order in which they were defined. For functions, the first parameter has the name of the function and represents the return value for the function.

Column name	Data type	Description
parm_type	SMALLINT	The type of parameter is one of the following: **0** Normal parameter (variable) **1** Result column - used with a procedure that returns result sets **2** SQLSTATE error value **3** SQLCODE error value **4** Return value from function
parm_mode_in	CHAR(1)	Indicates whether the parameter supplies a value to the procedure or function (IN or INOUT parameters).
parm_mode_out	CHAR(1)	Indicates whether the parameter returns a value from the procedure or function (OUT or INOUT parameters) or columns in the RESULT clause.
domain_id	SMALLINT	Identifies the data type for the parameter, by the data type number listed in the SYSDOMAIN system view.
width	BIGINT	Contains the length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.
scale	SMALLINT	For numeric data types, the number of digits after the decimal point. For all other data types, the value of this column is 1.
user_type	SMALLINT	The user type of the parameter, if applicable.
parm_name	CHAR(128)	The name of the parameter.
"default"	LONG VARCHAR	Default value of the parameter. Provided for informational purposes only.
remarks	LONG VARCHAR	Always returns NULL. Provided to allow the use of previous versions of ODBC drivers with newer personal database servers.
base_type_str	VARCHAR(32767)	The annotated type string representing the physical type of the parameter.

Remarks

The SYSPROCPARM system view is updated when a procedure or function is created or altered, including using the ALTER PROCEDURE...RECOMPILE statement.

Additionally, SYSPROCPARM is updated whenever a checkpoint is run if the out-of-date procedure or function meets the following conditions:

- The procedure or function has been referenced since it was altered.
- The procedure either has a RESULT clause or is not a recursive procedure with calls nested ten deep to other procedures that do not have RESULT clauses.

Related Information

Outdated Result Sets and Parameters in the SYSPROCPARM System View

1.7.1.49 SYSPROCPERM System View

Each row of the SYSPROCPERM system view describes a user who has been granted EXECUTE privilege on a procedure. The underlying system table for this view is ISYSPROCPERM.

Column name	Data type	Description
proc_id	UNSIGNED INT	The procedure number uniquely identifies the procedure for which EXECUTE privilege has been granted.
grantee	UNSIGNED INT	The user number of the privilege grantee.

1.7.1.50 SYSPROXYTAB System View

Each row of the SYSPROXYTAB system view describes the remote parameters of one proxy table. The underlying system table for this view is ISYSPROXYTAB.

Column name	Data type	Description
table_object_id	UNSIGNED BIGINT	The object ID of the proxy table.
existing_obj	CHAR(1)	Indicates whether the proxy table previously existed on the remote server.
srvid	UNSIGNED INT	The unique ID for the remote server associated with the proxy table.
remote_location	LONG VARCHAR	The location of the proxy table on the remote server.

Column name	Data type	Description
location_escape_char	CHAR(1)	The escape character that is used to escape the location delimiter.

1.7.1.51 SYSPUBLICATION System View

Each row in the SYSPUBLICATION system view describes a publication. The underlying system table for this view is ISYSPUBLICATION.

Column name	Data type	Description
publication_id	UNSIGNED INT	A number uniquely identifying the publication.
object_id	UNSIGNED BIGINT	The internal ID for the publication, uniquely identifying it in the database.
creator	UNSIGNED INT	The owner of the publication.
publication_name	CHAR(128)	The name of the publication.
remarks	LONG VARCHAR	Remarks about the publication. This value is stored in the ISYSREMARK system table.
type	CHAR(1)	This column is deprecated.
sync_type	UNSIGNED INT	The type of synchronization for the publication. Values include: **0 (logscan)** This is a regular publication that uses the transaction log to upload all relevant data that has changed since the last upload. **1 (scripted upload)** For this publication, the transaction log is ignored and the upload is defined by the user using stored procedures. Information about the stored procedures is stored in the ISYSSYNCSCRIPT system table. **2 (download only)** This is a download-only publication; no data is uploaded.

Related Information

Scripted Upload

SYSSYNCSCRIPT System View [page 1950]

1.7.1.52 SYSREMARK System View

Each row in the SYSREMARK system view describes a remark (or comment) for an object. The underlying system table for this view is ISYSREMARK.

Column	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the object that has an associated remark.
remarks	LONG VARCHAR	The remark or comment associated with the object.

1.7.1.53 SYSREMOTEOPTION System View

Each row in the SYSREMOTEOPTION system view describes the value of a message link parameter. The underlying system table for this view is ISYSREMOTEOPTION.

Some columns in this view contain potentially sensitive data. The SYSREMOTEOPTION2 view provides public access to the data in this view except for the potentially sensitive columns.

Column	Data type	Description
option_id	UNSIGNED INT	An identification number for the message link parameter.
user_id	UNSIGNED INT	The user ID for which the parameter is set.
setting	VARCHAR(255)	The value of the message link parameter.

1.7.1.54 SYSREMOTEOPTIONTYPE System View

Each row in the SYSREMOTEOPTIONTYPE system view describes one of the message link parameters. The underlying system table for this view is ISYSREMOTEOPTIONTYPE.

Column	Data type	Description
option_id	UNSIGNED INT	An identification number for the message link parameter.
type_id	SMALLINT	An identification number for the message type that uses the parameter.
option	VARCHAR(128)	The name of the message link parameter.

Constraints on underlying system table

```
PRIMARY KEY (option_id)
```

```
FOREIGN KEY (type_id) REFERENCES SYS.ISYSREMOTETYPE (type_id)
```

1.7.1.55 SYSREMOTETYPE System View

The SYSREMOTETYPE system view contains information about remote tables. The underlying system table for this view is ISYSREMOTETYPE.

Column name	Data type	Description
type_id	SMALLINT	Identifies which of the message systems is to be used to send messages to the user.
object_id	UNSIGNED BIGINT	The internal ID for the remote type, uniquely identifying it in the database.
type_name	CHAR(128)	The name of the message system.
publisher_address	LONG VARCHAR	The address of the remote database publisher.
remarks	LONG VARCHAR	Remarks about the remote type. This value is stored in the ISYSREMARK system table.

1.7.1.56 SYSREMOTEUSER System View

Each row in the SYSREMOTEUSER system view describes a user ID with the REMOTE system privilege (a subscriber), together with the status of messages that were sent to and from that user. The underlying system table for this view is ISYSREMOTEUSER.

Column name	Data type	Description
user_id	UNSIGNED INT	The user number of the user with REMOTE privilege.
consolidate	CHAR(1)	Indicates whether the user was granted CONSOLIDATE privilege (Y) or REMOTE privileges (N).
type_id	SMALLINT	Identifies which of the message systems is used to send messages to the user.

Column name	Data type	Description
address	LONG VARCHAR	The address to which messages are to be sent. The address must be appropriate for the address_type.
frequency	CHAR(1)	How frequently messages are sent.
send_time	TIME	The next time messages are to be sent to this user.
log_send	UNSIGNED BIGINT	Messages are sent only to subscribers for whom log_send is greater than log_sent.
time_sent	TIMESTAMP	The local time the most recent message was sent to this subscriber.
log_sent	UNSIGNED BIGINT	The log offset for the most recently sent operation.
confirm_sent	UNSIGNED BIGINT	The log offset for the most recently confirmed operation from this subscriber.
send_count	INTEGER	How many messages have been sent.
resend_count	INTEGER	Counter to ensure that messages are applied only once at the subscriber database.
time_received	TIMESTAMP	The local time when the most recent message was received from this subscriber.
log_received	UNSIGNED BIGINT	The log offset in the database of the subscriber for the operation that was most recently received at the current database.
confirm_received	UNSIGNED BIGINT	The log offset in the database of the subscriber for the most recent operation for which a confirmation message has been sent.
receive_count	INTEGER	How many messages have been received.
rereceive_count	INTEGER	Counter to ensure that messages are applied only once at the current database.
time_sent_utc	TIMESTAMP WITH TIME ZONE	The UTC time the most recent message was sent to this subscriber.
time_received_utc	TIMESTAMP WITH TIME ZONE	The UTC time when the most recent message was received from this subscriber.

1.7.1.57 SYSROLEGRANT System View

The SYSROLEGRANT system view stores information about role membership and type of membership. The underlying system table for this view is ISYSROLEGRANT.

Column name	Data type	Description
grant_id	UNSIGNED INT	ID used to identify each GRANT statement.
role_id	UNSIGNED INT	ID of the role being granted, as per ISYSUSER.
grantee	UNISIGNED INT	ID of the user being granted the role, as per ISYSUSER.
grant_type	TINYINT	Describes type of grant using 3 digits. The first bit from the right is whether privilege has been granted. The second digit is whether administration rights have been given. The third digit is whether system privileges are inheritable. **001** Privilege granted, with no inheritance, and no administration rights. Applicable only for legacy non-inheritable authorities except DBA and REMOTE DBA. **101** Privilege granted, with inheritance, but no administration rights. **110** Only administration rights have been granted. **111** Privilege granted, with inheritance, and with administration rights.

Column name	Data type	Description
grant_scope	TINYINT	Used by SET USER and CHANGE PASSWORD to set the scope of the grant. Values can be one or more of the following: **1** ANY **2** User list **4** Role list
grantor	CHAR(128)	The name of the grantor.

1.7.1.58 SYSROLEGRANTEXT System View

When you grant the SET USER and the CHANGE PASSWORD system privileges, you can specify a list of users or roles that the grantee can grant them to.

The SYSROLEGRANTEXT system view stores information about which users and roles the grantee can grant SET USER and CHANGE PASSWORD system privileges to.

The underlying system table for this view is ISYSROLEGRANTEXT.

Column name	Data type	Description
grant_id	UNSIGNED INT	ID used to identify each GRANT statement.
user_id	UNSIGNED INT	The user_ids specified in `user-list` or `role-list` in a particular extended grant

Remarks

When you grant or revoke the SET USER or CHANGE PASSWORD privilege, either with the `user-list` option or with ANY WITH ROLES `role-list` option, this view is updated with the values from the extended syntax.

1.7.1.59 SYSSCHEDULE System View

Each row in the SYSSCHEDULE system view describes a time at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT. The underlying system table for this view is ISYSSCHEDULE.

Column name	Data type	Description
event_id	UNSIGNED INT	The unique number assigned to each event.
sched_name	VARCHAR(128)	The name associated with the schedule for the event.
recurring	TINYINT	Indicates if the schedule is repeating.
start_time	TIME	The schedule start time.
stop_time	TIME	The schedule stop time if BETWEEN was used.
start_date	DATE	The first date on which the event is scheduled to execute.
days_of_week	TINYINT	A bit mask indicating the days of the week on which the event is scheduled: • x01 = Sunday • x02 = Monday • x04 = Tuesday • x08 = Wednesday • x10 = Thursday • x20 = Friday • x40 = Saturday
days_of_month	UNSIGNED INT	A bit mask indicating the days of the month on which the event is scheduled. Some examples include: • x01 = first day • x02 = second day • x40000000 = 31st day • x80000000 = last day of month
interval_units	CHAR(10)	The interval unit specified by EVERY: • HH = hours • NN = minutes • SS = seconds
interval_amt	INTEGER	The period specified by EVERY.

1.7.1.60 SYSSEQUENCE System View

The SYSSEQUENCE system view contains one row for each user-defined sequence. The underlying system table for this view is ISYSSEQUENCE.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The unique number assigned to each sequence.
owner	UNSIGNED INT	The owner of the sequence.
min_value	BIGINT	The minimum value allowed for the sequence.
max_value	BIGINT	The maximum value allowed for the sequence.
increment_by	BIGINT	The increment value for the sequence.
start_with	BIGINT	The starting value for the sequence.
cache	UNSIGNED INT	The number of sequence values to preallocate in memory for faster access. A value of 0 indicates that values are not to be preallocated
cycle	TINYINT	Whether values should continue to be generated after the maximum or minimum value is reached.
resume_at	BIGINT	The RESTART WITH value specified by the ALTER SEQUENCE statement. The value is NULL if no ALTER RESTART WITH statement has been executed.
sequence_name	CHAR(128)	The name of the sequence.

1.7.1.61 SYSSEQUENCEPERM System View

The SYSSEQUENCEPERM system view records the privileges that users or groups hold on sequences. The underlying system table for this view is ISYSSEQUENCEPERM.

Column name	Data type	Description
sequence_id	UNSIGNED BIGINT	The unique number assigned to each sequence.
grantee	UNSIGNED INT	The ID of the user or group with privileges to alter or drop the sequence.
grantor	UNSIGNED INT	The ID of the user who granted the privileges for the sequence.
privilege_type	SMALLINT	The type of privileges granted to the user or group on the sequence.

1.7.1.62 SYSSERVER System View

Each row in the SYSSERVER system view describes a remote server. The underlying system table for this view is ISYSSERVER.

> **i** Note
>
> Previous versions of the catalog contained a SYSSERVERS system table. That table has been renamed to be ISYSSERVER (without an 'S'), and is the underlying table for this view.

Column name	Data type	Description
srvid	UNSIGNED INT	An identifier for the remote server.
srvname	VARCHAR(128)	The name of the remote server.
srvclass	LONG VARCHAR	The server class, as specified in the CREATE SERVER statement.
srvinfo	LONG VARCHAR	Server information.
srvreadonly	CHAR(1)	Whether the server is read-only.

1.7.1.63 SYSSOURCE System View

Each row in the SYSSOURCE system view contains the source code, if applicable, for an object listed in the SYSOBJECT system view. The underlying system table for this view is ISYSSOURCE.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the object whose source code is being defined.
source	LONG VARCHAR	This column contains the original source code for the object if the pre-serve_source_format database option is On when the object was created.

Related Information

preserve_source_format Option

1.7.1.64 SYSSPATIALREFERENCESYSTEM System View

Each row of the SYSSPATIALREFERENCESYSTEM system view describes an SRS defined in the database. The underlying system table for this view is ISYSSPATIALREFERENCESYSTEM.

This view offers slightly different amount of information than the ST_SPATIAL_REFERENCE_SYSTEMS system view.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	For system use only.
owner	UNSIGNED INT	The owner of the SRS.
srs_name	CHAR(128)	The name of the SRS.
srs_id	INTEGER	The numeric identifier (SRID) for the spatial reference system.
round_earth	CHAR(1)	Whether the SRS type is ROUND EARTH (Y) or PLANAR (N).
axis_order	CHAR(12)	Describes how the database server interprets points with regards to latitude and longitude (for example when using the ST_Lat and ST_Long methods). For non-geographic spatial reference systems, the axis order is x/y/z/m. For geographic spatial reference systems, the default axis order is long/lat/z/m; lat/long/z/m is also supported.
snap_to_grid	DOUBLE	Defines the size of the grid the database server uses when performing calculations.
tolerance	DOUBLE	Defines the precision to use when comparing points.
semi_major_axis	DOUBLE	Distance from center of the ellipsoid to the equator for a ROUND EARTH SRS.
semi_minor_axis	DOUBLE	Distance from center of the ellipsoid to the poles for a ROUND EARTH SRS.

Column name	Data type	Description
inv_flattening	DOUBLE	The inverse flattening used for the ellipsoid in a ROUND EARTH SRS. Inverse flattening (f) is a mathematical value that defines the degree of squashing of the pole of a spheroid towards its equator. The value ranges from no flattening (a perfect circle) to complete flattening (a straight line). Inverse flattening is the value of 1/f, as follows: `1/f = (semi_major_axis) / (semi_major_axis - semi_minor_axis)`
min_x	DOUBLE	The minimum x value allowed in coordinates.
max_x	DOUBLE	The maximum x value allowed in coordinates.
min_y	DOUBLE	The minimum y value allowed in coordinates.
max_y	DOUBLE	The maximum y value allowed in coordinates.
min_z	DOUBLE	The minimum z value allowed in coordinates.
max_z	DOUBLE	The maximum z value allowed in coordinates.
min_m	DOUBLE	The minimum m value allowed in coordinates.
max_m	DOUBLE	The maximum m value allowed in coordinates.
organization	LONG VARCHAR	The name of the organization that created the coordinate system used by the spatial reference system.
organization_coordsys_id	INTEGER	The ID given to the coordinate system by the organization that created it.

Column name	Data type	Description
srs_type	CHAR(11)	The type of SRS as defined by the SQL/MM standard. Values can be one of: **GEOGRAPHIC** This is for SRSs based on georeferenced coordinate systems with axes of latitude, longitude (and elevation). These SRSs are of type PLANAR or ROUND EARTH. **PROJECTED** This is for SRSs based on georeferenced coordinate systems that do not have axes of latitude and longitude. These SRSs are of type PLANAR. **ENGINEERING** This is for SRSs based on non-georeferenced coordinate systems. These SRSs are of type PLANAR. **GEOCENTRIC** Unsupported. **COMPOUND** Unsupported. **VERTICAL** Unsupported. If srs_type is empty, the type is unspecified.
linear_unit_of_measure	UNSIGNED BIGINT	The linear unit of measure used by the spatial reference system.
angular_unit_of_measure	UNSIGNED BIGINT	The angular unit of measure used by the spatial reference system.
count_in_use	UNSIGNED BIGINT	For internal use only.
polygon_format	LONG VARCHAR	The orientation of the rings in a polygon. One of CounterClockwise, ClockWise, or EvenOdd.
storage_format	LONG VARCHAR	Whether the data is stored in normalized format (Internal), unnormalized format (Original), or both (Mixed).
definition	LONG VARCHAR	The WKT definition of the spatial reference system in the format defined by the OGC standard.

Column name	Data type	Description
transform_definition	LONG VARCHAR	Transform definition settings for use when transforming data from this SRS to another.

1.7.1.65 SYSSQLSERVERTYPE System View

The SYSSQLSERVERTYPE system view contains information relating to compatibility with Adaptive Server Enterprise. The underlying system table for this view is ISYSSQLSERVERTYPE.

Column name	Data type	Description
ss_user_type	SMALLINT	The Adaptive Server Enterprise user type.
ss_domain_id	SMALLINT	The Adaptive Server Enterprise domain ID.
ss_type_name	VARCHAR (30)	The Adaptive Server Enterprise type name.
primary_sa_domain_id	SMALLINT	The corresponding SQL Anywhere primary domain ID.
primary_sa_user_type	SMALLINT	The corresponding SQL Anywhere primary user type.

1.7.1.66 SYSSUBSCRIPTION System View

Each row in the SYSSUBSCRIPTION system view describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication. The underlying system table for this view is ISYSSUBSCRIPTION.

Column name	Data type	Description
publication_id	UNSIGNED INT	The identifier for the publication to which the user ID is subscribed.
user_id	UNSIGNED INT	The ID of the user who is subscribed to the publication.
subscribe_by	CHAR(128)	The value of the SUBSCRIBE BY expression, if any, for the subscription.
created	UNSIGNED BIGINT	The offset in the transaction log at which the subscription was created.
started	UNSIGNED BIGINT	The offset in the transaction log at which the subscription was started.

1.7.1.67 SYSSYNC System View

The SYSSYNC system view contains information relating to synchronization.

The "option" and server_connect columns of the underlying table, ISYSSYNC, contain sensitive information such as passwords. You must have the SELECT ANY TABLE and ACCESS USER PASSWORD system privileges to select from this view. The SYSSYNC2 consolidated view provides public access to the same data without the sensitive data.

The underlying system table for this view is ISYSSYNC.

Column name	Data type	Description
sync_id	UNSIGNED INT	A number that uniquely identifies the row.
type	CHAR(1)	This value is always D.
publication_id	UNSIGNED INT	A publication_id found in the SYSPUBLICATION system view.
progress	UNSIGNED BIGINT	The log offset of the last successful upload.
site_name	CHAR(128)	A user name.
"option"	LONG VARCHAR	Synchronization options.
server_connect	LONG VARCHAR	The address or URL of the server.
server_conn_type	LONG VARCHAR	The communication protocol, such as TCP/IP, to use when synchronizing.
last_download_time	TIMESTAMP	Indicates the last time a download stream was received from the server.
last_upload_time	TIMESTAMP	Indicates the last time (measured at the server) that information was successfully uploaded. The default is jan-1-1900.
created	UNSIGNED BIGINT	The log offset at which the subscription was created.
log_sent	UNSIGNED BIGINT	The log progress up to which information has been uploaded. It is not necessary that an acknowledgment of the upload be received for the entry in this column to be updated.
generation_number	INTEGER	For file-base downloads, the last generation number received for this subscription. The default is 0.
extended_state	VARCHAR(1024)	For internal use only.

Column name	Data type	Description
script_version	CHAR(128)	Indicates the script version used by the CREATE and ALTER SYNCHRONIZATION SUBSCRIPTION statements and the START SYNCHRONIZATION SCHEMA CHANGE statement.
subscription_name	CHAR (128)	The name of the subscription.
server_protocol	UNSIGNED BIGINT	For internal use only. Contains a value used internally to identify the version of the synchronization server.

Related Information

SYSSYNC2 Consolidated View [page 1992]

1.7.1.68 SYSSYNCPROFILE System View

The SYSSYNCPROFILE system view contains information relating to synchronization profiles for MobiLink synchronization.

You must have the SELECT ANY TABLE and ACCESS USER PASSWORD system privileges to access this view.

The underlying system table for this view is ISYSSYNCPROFILE.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The object ID of the sync profile.
profile_name	CHAR(128)	The name of the sync profile.
profile_defn	LONG VARCHAR	The definition for the syn profile.

1.7.1.69 SYSSYNCPROFILE2 System View

The SYSSYNCPROFILE2 system view contains information relating to synchronization profiles for MobiLink synchronization.

The underlying system table for this view is ISYSSYNCPROFILE.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The object ID of the sync profile.
profile_name	CHAR(128)	The name of the sync profile.

1.7.1.70 SYSSYNCSCRIPT System View

Each row in the SYSSYNCSCRIPT system view identifies a stored procedure for scripted upload. This view is almost identical to the SYSSYNCSCRIPTS view, except that the values in this view are in their raw format.

The underlying system table for this view is ISYSSYNCSCRIPT.

Column name	Data type	Description
pub_object_id	UNSIGNED BIGINT	The object ID of the publication to which the script belongs.
table_object_id	UNSIGNED BIGINT	The object ID of the table to which the script applies.
type	UNSIGNED INT	The type of upload procedure.
proc_object_id	UNSIGNED BIGINT	The object ID of the stored procedure to use for the publication.

Related Information

Scripted Upload
SYSSYNCSCRIPTS Consolidated View [page 1994]
SYSPROCEDURE System View [page 1931]
SYSPUBLICATION System View [page 1935]

1.7.1.71 SYSTAB System View

Each row of the SYSTAB system view describes one table or view in the database. Additional information for views can be found in the SYSVIEW system view. The underlying system table for this view is ISYSTAB.

Column name	Data type	Description
table_id	UNSIGNED INT	Each table is assigned a unique number (the table number).
dbspace_id	SMALLINT	A value indicating which dbspace contains the table.
count	UNSIGNED BIGINT	The number of rows in the table or materialized view. This value is updated during each successful checkpoint. This number is used to optimize database access. The count is always 0 for a non-materialized view or remote table.
creator	UNSIGNED INT	The user number of the owner of the table or view.

Column name	Data type	Description
table_page_count	INTEGER	The total number of main pages used by the underlying table.
ext_page_count	INTEGER	The total number of extension pages used by the underlying table.
commit_action	INTEGER	For global temporary tables, 0 indicates that the ON COMMIT PRESERVE ROWS clause was specified when the table was created, 1 indicates that the ON COMMIT DELETE ROWS clause was specified when the table was created (the default behavior for temporary tables), and 3 indicates that the NOT TRANSACTIONAL clause was specified when the table was created. For non-temporary tables, commit_action is always 0.
share_type	INTEGER	For global temporary tables, 4 indicates that the SHARE BY ALL clause was specified when the table was created, and 5 indicates that the SHARE BY ALL clause was *not* specified when the table was created. For non-temporary tables, share_type is always 5 because the SHARE BY ALL clause cannot be specified when creating non-temporary tables.
object_id	UNSIGNED BIGINT	The object ID of the table.
last_modified_at	TIMESTAMP	The local time at which the data in the table was last modified. This column is only updated at checkpoint time.
table_name	CHAR(128)	The name of the table or view. One creator cannot have two tables or views with the same name.

Column name	Data type	Description
table_type	TINYINT	The type of table or view. Values include: **1** Base table **2** Materialized view **3** Global temporary table **4** Local temporary table **5** Text index base table **6** Text index global temporary table **21** View
replicate	CHAR(1)	This value is for internal use only.
server_type	TINYINT	The location of the data for the underlying table. Values include: **1** Local server **3** Remote server
tab_page_list	LONG VARBIT	For internal use only. The set of pages that contain information for the table, expressed as a bitmap.
ext_page_list	LONG VARBIT	For internal use only. The set of pages that contain row extensions and large object (LOB) pages for the table, expressed as a bitmap.
pct_free	UNSIGNED INT	The PCT_FREE specification for the table, if one has been specified; otherwise, NULL.
clustered_index_id	UNSIGNED INT	The ID of the clustered index for the table. If none of the indexes are clustered, then this field is NULL.
encrypted	CHAR(1)	Whether the table or materialized view is encrypted.

Column name	Data type	Description
last_modified_tsn	UNSIGNED BIGINT	A sequence number assigned to the transaction that modified the table. This column is only updated at checkpoint time.
current_schema	UNSIGNED INT	The current schema version of the table.
file_id	SMALLINT	DEPRECATED. This column is present in SYSVIEW, but not in the underlying system table ISYSTAB. The contents of this column is the same as dbspace_id and is provided for compatibility. Use dbspace_id instead.
table_type_str	CHAR(13)	Readable value for table_type. Values include: **BASE** Base table **MAT VIEW** Materialized view **GBL TEMP** Global temporary table **VIEW** View
last_modified_at_utc	TIMESTAMP WITH TIME ZONE	The UTC time at which the data in the table was last modified. This column is only updated at checkpoint time.

Related Information

SYSVIEW System View [page 1970]

1.7.1.72 SYSTABCOL System View

The SYSTABCOL system view contains one row for each column of each table and view in the database. The underlying system table for this view is ISYSTABCOL.

Column name	Data type	Description
table_id	UNSIGNED INT	The object ID of the table or view to which the column belongs.

Column name	Data type	Description
column_id	UNSIGNED INT	The ID of the column. For each table, column numbering starts at 1. The column_id value determines the order of columns in the result set when SELECT * is used. It also determines the column order for an INSERT statement when a list of column names is not provided.
domain_id	SMALLINT	The data type for the column, indicated by a data type number listed in the SYSDOMAIN system view.
nulls	CHAR(1)	Indicates whether NULL values are allowed in the column.
width	BIGINT	The length of a string column, the precision of numeric columns, or the number of bytes of storage for any other data type.
scale	SMALLINT	The number of digits after the decimal point for NUMERIC or DECIMAL data type columns. For string columns, a value of 1 indicates character-length semantics and 0 indicates byte-length semantics.
object_id	UNSIGNED BIGINT	The object ID of the table column.
max_identity	BIGINT	The largest value of the column, if it is an AUTOINCREMENT, IDENTITY, or GLOBAL AUTOINCREMENT column.
column_name	CHAR(128)	The name of the column.
"default"	LONG VARCHAR	The default value for the column. This value, if specified, is only used when an INSERT statement does not specify a value for the column.
user_type	SMALLINT	The data type, if the column is defined using a user-defined data type.
column_type	CHAR(1)	The type of column (C=computed column, and R=other columns).
compressed	TINYINT	Whether this column is stored in a compressed format.
collect_stats	TINYINT	Whether the system automatically collects and updates statistics on this column.

Column name	Data type	Description
inline_max	SMALLINT	The maximum number of bytes of a BLOB to store in a row. A NULL value indicates that either the default value has been applied, or that the column is not a character or binary type. A non-NULL inline_max value corresponds to the INLINE value specified for the column using the CREATE TABLE or ALTER TABLE statement.
inline_long	SMALLINT	The number of duplicate bytes of a BLOB to store in a row if the BLOB size exceeds the inline_max value. A NULL value indicates that either the default value has been applied, or that the column is not a character or binary type. A non-NULL inline_long value corresponds to the PREFIX value specified for the column using the CREATE TABLE or ALTER TABLE statement.
lob_index	TINYINT	Whether to build indexes on BLOB values in the column that exceed an internal threshold size (approximately eight database pages). A NULL value indicates either that the default is applied, or that the column is not BLOB type. A value of 1 indicates that indexes will be built. A value of 0 indicates that no indexes will be built. A non-NULL lob_index value corresponds to whether INDEX or NO INDEX was specified for the column using the CREATE TABLE or ALTER TABLE statement.
base_type_str	VARCHAR(32,767)	The annotated type string representing the physical type of the column.
nonmaterialized_value	LONG BINARY	Internal use only.
start_schema	UNSIGNED INT	The first version of the table schema in which this column exists.

Related Information

CREATE TABLE Statement [page 1002]

1.7.1.73 SYSTABLEPERM System View

Privileges granted on tables and views by the GRANT statement are stored in the SYSTABLEPERM system view. Each row in this view corresponds to one table, one user ID granting the privilege (grantor) and one user ID granted the privilege (grantee). The underlying system table for this view is ISYSTABLEPERM.

Column name	Data type	Description
stable_id	UNSIGNED INT	The table number of the table or view to which the privileges apply.
grantee	UNSIGNED INT	The user number of the user ID receiving the privilege.
grantor	UNSIGNED INT	The user number of the user ID granting the privilege.
selectauth	CHAR(1)	Indicates whether SELECT privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
insertauth	CHAR(1)	Indicates whether INSERT privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
deleteauth	CHAR(1)	Indicates whether DELETE privileges has been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
updateauth	CHAR(1)	Indicates whether UPDATE privileges have been granted for all columns in the table. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
updatecols	CHAR(1)	Indicates whether UPDATE privileges have only been granted for some of the columns in the underlying table. If updatecols has the value Y, there will be one or more rows in the SYSCOLPERM system view granting update privileges for the columns.
alterauth	CHAR(1)	Indicates whether ALTER privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.

Column name	Data type	Description
referenceauth	CHAR(1)	Indicates whether REFERENCE privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
loadauth	CHAR(1)	Indicates whether LOAD privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
truncateauth	CHAR(1)	Indicates whether TRUNCATE privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
loadauth	CHAR(1)	Indicates whether LOAD privileges has been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.
truncateauth	CHAR(1)	Indicates whether TRUNCATE privileges has been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.

Remarks

There are several types of privileges that can be granted. Each privilege can have one of the following three values.

N

No, the grantee has not been granted this privilege by the grantor.

Y

Yes, the grantee has been given this privilege by the grantor.

G

The grantee has been given this privilege and can grant the same privilege to another user.

> **i** Note
>
> The grantee might have been given the privilege for the same table by another grantor. If so, this information would be found in a different row of the SYSTABLEPERM system view.

Constraints on underlying system table

```
PRIMARY KEY (stable_id, grantee, grantor)
```

```
FOREIGN KEY (stable_id) REFERENCES SYS.ISYSTAB (table_id)
```

```
FOREIGN KEY (grantor) REFERENCES SYS.ISYSUSER (user_id)
```

```
FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)
```

Related Information

GRANT Statement [page 1193]

1.7.1.74 SYSTEXTCONFIG System View

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	The object ID for the text configuration object.
creator	UNSIGNED INT	The creator of the text configuration object.
term_breaker	TINYINT	The algorithm used to separate a string into terms or words. Values are 0 for GENERIC and 1 for NGRAM. With GENERIC, any string of one or more alphanumeric characters separated by non-alphanumerics are treated as a term. NGRAM is for approximate matching or for documents that do not use a white-space to separate terms.
stemmer	TINYINT	For internal use only.
min_term_length	TINYINT	The minimum length, in characters, allowed for a term. Terms that are shorter than min_term_length are ignored. The MINIMUM TERM LENGTH setting is only meaningful for the GENERIC term breaker. For NGRAM text indexes, the setting is ignored.

Column name	Data type	Description
max_term_length	TINYINT	For GENERIC text indexes, the maximum length, in characters, allowed for a term. Terms that are longer than max_term_length are ignored. For NGRAM text indexes, this is the length of the n-grams into which terms are broken.
collation	CHAR(128)	For internal use only.
text_config_name	CHAR(128)	The name of the text configuration object.
prefilter	LONG VARCHAR	The function and library name for an external prefilter library.
postfilter	LONG VARCHAR	For internal use only.
char_stoplist	LONG VARCHAR	Terms to ignore when performing a full text search on CHAR columns. These terms are also omitted from text indexes. This column is used when the text configuration object is created from default_char.
nchar_stoplist	LONG NVARCHAR	Terms to ignore when performing a full text search on NCHAR columns. These terms are also omitted from text indexes. This column is used when the text configuration object is created from default_nchar.
external_term_breaker	LONG VARCHAR	The function and library name for an external term breaker library.

Related Information

What to Specify When Creating or Altering Text Configuration Objects
Full Text Search

1.7.1.75 SYSTEXTIDX System View

Each row in the SYSTEXTIDX system view describes one text index. The underlying system table for this view is ISYSTEXTIDX.

Column name	Data type	Description
index_id	UNSIGNED BIGINT	The object ID of the text index in SYSIDX.
sequence	UNSIGNED INT	For internal use only.
status	UNSIGNED INT	For internal use only.
text_config	UNSIGNED BIGINT	The object ID of the text configuration object in SYSTEXTCONFIG.
next_handle	UNSIGNED INT	For internal use only.
last_handle	UNSIGNED INT	For internal use only.
deleted_length	UNSIGNED BIGINT	The total size of deleted indexed values in the text index.
pending_length	UNSIGNED BIGINT	The total size of indexed values that will be added to the text index at the next refresh.
refresh_type	TINYINT	The type of refresh. One of: **1** MANUAL **2** AUTO **3** IMMEDIATE
refresh_interval	UNSIGNED INT	The AUTO REFRESH interval, in minutes.
last_refresh	TIMESTAMP	The local time of the last refresh.
last_refresh_utc	TIMESTAMP WITH TIME ZONE	The UTC time of the last refresh.

1.7.1.76 SYSTEXTIDXTAB System View

Each row in the SYSTEXTIDXTAB system view describes a generated table that is part of a text index. The underlying system table for this view is ISYSTEXTIDXTAB.

Column name	Data type	Description
index_id	UNSIGNED BIGINT	For internal use only.
sequence	UNSIGNED INT	For internal use only.

Column name	Data type	Description
table_type	UNSIGNED INT	For internal use only.
table_id	UNSIGNED INT	For internal use only.

Related Information

Full Text Search

1.7.1.77 SYSTIMEZONE System View

Each row in the SYSTIMEZONE system view describes one time zone. The underlying system table for this view is ISYSTIMEZONE.

Column name	Data type	Description
timezone_id	UNSIGNED BIGINT	The object ID of the time zone in SYSTIMEZONE.
name	VARCHAR (128)	The name of the time zone.
offset	INTEGER	The offset for the time zone.
dst_offset	INTEGER	The offset for daylight savings time.
start_dst_month	INTEGER	The month that daylight savings time begins.
start_dst_day	VARCHAR (128)	The day that daylight savings time begins.
start_dst_time	INTEGER	The time that daylight savings time begins.
end_dst_month	INTEGER	The month that daylight savings time ends.
end_dst_day	VARCHAR (128)	The day that daylight savings time ends.
end_dst_time	INTEGER	The time that daylight savings time ends.
alias_id	UNSIGNED BIGINT	Internal use only.

Remarks

The underlying table for this view, ISYSTIMEZONE, is not populated for new databases. To populate ISYSTIMEZONE, execute one of the following statements:

CREATE TIME ZONE statement
ALTER TIME ZONE statement

Related Information

ALTER TIME ZONE Statement [page 764]
COMMENT Statement [page 808]
CREATE TIME ZONE Statement [page 1033]
DROP TIME ZONE Statement [page 1138]
time_zone Option

1.7.1.78 SYSTRIGGER System View

Each row in the SYSTRIGGER system view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE). The underlying system table for this view is ISYSTRIGGER.

Column name	Data type	Description
trigger_id	UNSIGNED INT	A unique number for the trigger in the SYSTRIGGER view.
table_id	UNSIGNED INT	The table ID of the table to which this trigger belongs.
object_id	UNSIGNED BIGINT	The object ID for the trigger in the database.

Column name	Data type	Description
event	CHAR(1)	The operation that causes the trigger to fire. **A** INSERT, DELETE **B** INSERT, UPDATE **C** UPDATE COLUMNS **D** DELETE **E** DELETE, UPDATE **I** INSERT **M** INSERT, DELETE, UPDATE **U** UPDATE
trigger_time	CHAR(1)	The time when the trigger fires relative to the event. **A** AFTER (row-level trigger) **B** BEFORE (row-level trigger) **I** INSTEAD OF (row-level trigger) **K** INSTEAD OF (statement-level trigger) **R** RESOLVE **S** AFTER (statement-level trigger)
trigger_order	SMALLINT	The order in which are fired when there are multiple triggers of the same type (insert, update, or delete) set to fire at the same time (applies to BEFORE or AFTER triggers only, only).

Column name	Data type	Description
foreign_table_id	UNSIGNED INT	The ID of the table containing a foreign key definition that has a referential triggered action (such as ON DELETE CASCADE). The foreign_table_id value reflects the value of ISYSIDX.table_id.
foreign_key_id	UNSIGNED INT	The ID of the foreign key for the table referenced by foreign_table_id. The foreign_key_id value reflects the value of ISYSIDX.index_id.
referential_action	CHAR(1)	The action defined by a foreign key. This single-character value corresponds to the action that was specified when the foreign key was created. **C** CASCADE **D** SET DEFAULT **N** SET NULL **R** RESTRICT
trigger_name	CHAR(128)	The name of the trigger. One table cannot have two triggers with the same name.
trigger_defn	LONG VARCHAR	The command that was used to create the trigger.
remarks	LONG VARCHAR	Remarks about the trigger. This value is stored in the ISYSREMARK system table.
source	LONG VARCHAR	The SQL source for the trigger. This value is stored in the ISYSSOURCE system table.

1.7.1.79 SYSTYPEMAP System View

The SYSTYPEMAP system view contains the compatibility mapping values for entries in the SYSSQLSERVERTYPE system view. The underlying system table for this view is ISYSTYPEMAP.

Column name	Data type	Description
ss_user_type	SMALLINT	Contains the Adaptive Server Enterprise user type.

Column name	Data type	Description
sa_domain_id	SMALLINT	Contains the corresponding SQL Anywhere domain_id.
sa_user_type	SMALLINT	Contains the corresponding SQL Anywhere user type.
nullable	CHAR(1)	Whether the type allows NULL values.

1.7.1.80 SYSUNITOFMEASURE System View

Each row of the SYSUNITOFMEASURE system view describes a unit of measure defined in the database. The underlying table for the SYSUNITOFMEASURE system view is the ISYSUNITOFMEASURE system table.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	For system use only.
owner	UNSIGNED INT	The owner of the unit of measure.
unit_name	CHAR(128)	The name of the unit of measure.
unit_type	CHAR(7)	Angular or linear.
conversion_factor	DOUBLE	The conversion factor for the unit of measure.

1.7.1.81 SYSUSER System View

Each row in the SYSUSER system view describes a user in the system.

Standalone roles are also stored in this view as well, but only the user_id, object_id, user_name, and user_type columns are meaningful for these roles. The underlying system table for this view is ISYSUSER.

Column name	Data type	Description
user_id	UNSIGNED INT	A unique identifier for the user assigned to the login policy.
object_id	UNSIGNED BIGINT	A unique identifier for the user in the database.
user_name	CHAR(128)	The login name for the user.
password	CHAR(3)	Whether a password is stored. Three asterisks (***) indicate that a password is stored. NULL indicates that no password is stored. You can return actual password hash values by querying the SYSUSERPASSWORD system view.

Column name	Data type	Description
login_policy_id	UNSIGNED BIGINT	A unique identifier for the login policy.
expire_password_on_login	TINYINT	A value that indicates if the password for the user expires at the next login.
password_creation_time	TIMESTAMP	The local time that the password was created for the user.
failed_login_attempts	UNSIGNED INT	The number of times that a user can fail to log in before the account is locked.
last_login_time	TIMESTAMP	The local time that the user last logged in.
user_type	TINYINT	A value that indicates whether the user is a regular user, or a role, or a user extended as a role. And whether the user, role, or extended role can be altered (mutable) or removed. Possible values: **1** Immutable system role. **5** Mutable system role **9** Immutable and removable system role. **12** Mutable and removable user. **13** Mutable and removable role. **14** Mutable and removable user extended as role.
user_dn	CHAR (1024)	An LDAP Distinguished Name (DN) identifier for the user that is unique within a domain and across domains. The DN is used to authenticate with an LDAP server.
user_dn_cached_at	TIMESTAMP	The time that the user_dn column was last cached. This value is used to determine whether to purge an old DN. Regardless of the database server local time zone, the value is stored in Coordinated Universal Time (UTC).
password_creation_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time that the password was created for the user.

Column name	Data type	Description
last_login_time_utc	TIMESTAMP WITH TIME ZONE	The UTC time that the user last logged in.
dual_password	CHAR(3)	Whether the second part of a dual password is stored for the user. Three asterisks (***) indicate that a second part is stored. NULL indicates that there is no second part. You can return actual password hash values by querying the SYSUSERPASSWORD system view.
lock_time	TIMESTAMP	Timestamp at which user was locked due to failed login attempts.

Related Information

SYSUSERPASSWORD System View [page 1967]
sp_sys_priv_role_info System Procedure [page 1836]
SYSLOGINPOLICY System View [page 1923]
SYSLOGINPOLICYOPTION System View [page 1923]

1.7.1.82 SYSUSERPASSWORD System View

Each row in the SYSUSERPASSWORD system view gives the object ID and password hash for a login account. The underlying system table for this view is ISYSUSER.

You must have the SELECT ANY TABLE and ACCESS USER PASSWORD system privileges to access this view.

Column	Data type	Description
object_id	UNSIGNED BIGINT	The internal ID for the login account.
password	BINARY(128)	The password hash for a login account.
dual_password	BINARY(128)	The password hash for part of a dual password.

Related Information

SYSUSER System View [page 1965]

1.7.1.83 SYSUSERMESSAGE System View

Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.

Previous versions of the catalog contained a SYSUSERMESSAGES system table. That table has been renamed to be ISYSUSERMESSAGE (without an 'S'), and is the underlying table for this view.

Column name	Data type	Description
error	INTEGER	A unique identifying number for the error condition.
uid	UNSIGNED INT	The user number that defined the message.
description	VARCHAR(255)	The message corresponding to the error condition.
langid	SMALLINT	Reserved.

1.7.1.84 SYSUSERTYPE System View

Each row in the SYSUSERTYPE system view holds a description of a user-defined data type. The underlying system table for this view is ISYSUSERTYPE.

Column name	Data type	Description
type_id	SMALLINT	A unique identifying number for the user-defined data type.
creator	UNSIGNED INT	The user number of the owner of the data type.
domain_id	SMALLINT	The data type on which this user defined data type is based, indicated by a data type number listed in the SYSDOMAIN system view.
nulls	CHAR(1)	Whether the user-defined data type allows nulls. Possible values are Y, N, or U. A value of U indicates that nullability is unspecified.
width	BIGINT	The length of a string column, the precision of a numeric column, or the number of bytes of storage for any other data type.
scale	SMALLINT	The number of digits after the decimal point for numeric data type columns, and zero for all other data types.
type_name	CHAR(128)	The name for the data type.
"default"	LONG VARCHAR	The default value for the data type.

Column name	Data type	Description
"check"	LONG VARCHAR	The CHECK condition for the data type.
base_type_str	VARCHAR(32767)	The annotated type string representing the physical type of the user type.

1.7.1.85 SYSDATABASEVARIABLE System View

Each row in the SYSDATABASEVARIABLE system view describes one database-scope variable in the database. The underlying system table for this view is ISYSDATABASEVARIABLE.

Column name	Data type	Description
variable_id	UNSIGNED INT	The ID of the database variable.
object_id	UNSIGNED BIGINT	The internal ID for the database-scope variable, uniquely identifying it in the database.
owner	UNSIGNED INT	The owner of the database variable.
variable_name	CHAR(128)	The name of the database variable.
domain_id	SMALLINT	The ID of the data type as listed in the SYSDOMAIN system view.
width	UNSIGNED INT	The length of a string the database variable can hold, the precision of numeric values for the column, or the number of bytes of storage needed for any other data type.
scale	SMALLINT	The number of digits after the decimal point for NUMERIC or DECIMAL data type variables. For a database variable containing a string, a value of 1 indicates character-length semantics and 0 indicates byte-length semantics.
user_type	SMALLINT	The data type of the database variable, or NULL if no type value is present.
initial_value	LONG BINARY	The initial value of the database variable. If no value is specified, NULL is used. If the initial value was set using an expression, the expression is evaluated at creation time and the resulting constant is stored in this column (not the expression).

Column name	Data type	Description
base_type_str	VARCHAR(32767)	The annotated type string representing the physical type of the database variable.
initial_value_string	LONG VARCHAR	The string representation of the initial value of the database variable.

Remarks

Updates to database-scope variable values, for example using the SET statement, do not persist after a database restart. Also, updated values are not reflected in this view; only the initial/default value is visible in this view.

Privileges

None.

Related Information

SQL Variables [page 123]
CREATE VARIABLE Statement [page 1047]

1.7.1.86 SYSVIEW System View

Each row in the SYSVIEW system view describes a view in the database. Additional information about views can also be found in the SYSTAB system view. The underlying system table for this view is ISYSVIEW.

Use the sa_materialized_view_info system procedure for a more readable format of the information for materialized views.

Column name	Data type	Description
view_object_id	UNSIGNED BIGINT	The object ID of the view.
view_def	LONG VARCHAR	The definition (query specification) of the view.
mv_build_type	TINYINT	Internal use only.

Column name	Data type	Description
mv_refresh_type	TINYINT	The refresh type defined for the view. Possible values are IMMEDIATE (1) and MANUAL (2).
mv_use_in_optimization	TINYINT	Whether the materialized view can be used during query optimization (0=cannot be used in optimization, 1=can be used in optimization)
mv_last_refreshed_at	TIMESTAMP	Indicates the local date and time that the materialized view was last refreshed.
mv_known_stale_at	TIMESTAMP	The local time at which the materialized view became stale. This value corresponds to the time at which one of the underlying base tables was detected as having changed. A value of 0 indicates that the view is either fresh, or that it has become stale but the database server has not marked it as such because the view has not been used since it became stale. Use the sa_materialized_view_info system procedure to determine the status of a materialized view.
mv_last_refreshed_tsn	UNSIGNED BIGINT	The sequence number assigned to the transaction that refreshed the materialized view.
mv_last_refreshed_at_utc	TIMESTAMP WITH TIME ZONE	Indicates the UTC date and time that the materialized view was last refreshed.
mv_known_stale_at_utc	TIMESTAMP WITH TIME ZONE	The UTC time at which the materialized view became stale. This value corresponds to the time at which one of the underlying base tables was detected as having changed. A value of 0 indicates that the view is either fresh, or that it has become stale but the database server has not marked it as such because the view has not been used since it became stale. Use the sa_materialized_view_info system procedure to determine the status of a materialized view. This column contains 0 when mv_last_refreshed_at is 0 and NULL when mv_last_refreshed_at is NULL.
check_option	CHAR(1)	Returns Y if the view uses WITH CHECK OPTION and N if it does not.

Remarks

When a materialized view is refreshed with SNAPSHOT isolation, mv_last_refreshed_at and mv_last_refreshed_tsn refer to the earliest transaction that modified any row used during the computation of the materialized view contents.

Related Information

Whether to Set Refresh Type to Manual or Immediate
Enabling or Disabling Optimizer Use of a Materialized View
sa_materialized_view_info System Procedure [page 1643]
SYSTAB System View [page 1950]
CREATE MATERIALIZED VIEW Statement [page 896]
REFRESH MATERIALIZED VIEW Statement [page 1321]
CREATE VIEW Statement [page 1051]

1.7.1.87 SYSWEBSERVICE System View

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

Column name	Data type	Description
service_id	UNSIGNED INT	A unique identifying number for the web service.
object_id	UNSIGNED BIGINT	The ID of the webservice.
service_name	CHAR(128)	The name assigned to the web service.
service_type	VARCHAR(40)	The type of the service; for example, RAW, HTTP, XML, SOAP, or DISH.
auth_required	CHAR(1)	Whether all requests must contain a valid user name and password.
secure_required	CHAR(1)	Whether insecure connections, such as HTTP, are to be accepted, or only secure connections, such as HTTPS.
url_path	CHAR(1)	Controls the interpretation of URLs.
user_id	UNSIGNED INT	If authentication is enabled, identifies the user, or group of users, that have permission to use the service. If authentication is disabled, specifies the account to use when processing requests.

Column name	Data type	Description
parameter	LONG VARCHAR	A prefix that identifies the SOAP services to be included in a DISH service.
statement	LONG VARCHAR	A SQL statement that is always executed in response to a request. If NULL, arbitrary statements contained in each request are executed instead. Ignored for services of type DISH.
remarks	LONG VARCHAR	Remarks about the webservice. This value is stored in the ISYSREMARK system table.
enabled	CHAR(1)	Indicates whether the web service is currently enabled or disabled (see CREATE SERVICE).

1.7.2 Consolidated Views

Consolidated views provide data in a form more frequently required by users.

For example, consolidated views often provide commonly needed joins. Consolidated views differ from system views in that they are not just a straight forward view of raw data in a underlying system table(s). For example, many of the columns in the system views are unintelligible ID values, whereas in the consolidated views, they are readable names.

In SQL Anywhere, the system owner (user SYS) owns the consolidated views.

In this section:

Each row in the SYSCATALOG view describes a system table.

SYSCOLAUTH Consolidated View [page 1981]
Each row in the SYSCOLAUTH view describes the set of privileges (UPDATE, SELECT, or REFERENCES) granted on a column.

SYSCOLSTATS Consolidated View [page 1982]
The SYSCOLSTATS view contains the column statistics that are stored as histograms and used by the optimizer.

SYSCOLUMNS Consolidated View [page 1983]
Each row in the SYSCOLUMNS view describes one column of each table and view in the catalog.

SYSFOREIGNKEYS Consolidated View [page 1983]
Each row in the SYSFOREIGNKEYS view describes one foreign key for each table in the catalog.

SYSINDEXES Consolidated View [page 1984]
Each row in the SYSINDEXES view describes one index in the database. As an alternative to this view, you could also use the SYSIDX and SYSIDXCOL system views.

SYSOPTIONS Consolidated View [page 1985]
Each row in the SYSOPTIONS view describes one option created using the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user define the default settings to be used for users that do not have their own setting.

SYSPROCAUTH Consolidated View [page 1986]
Each row in the SYSPROCAUTH view describes a set of privileges granted on a procedure. As an alternative, you can also use the SYSPROCPERM system view.

SYSPROCPARMS Consolidated View [page 1986]
Each row in the SYSPROCPARMS view describes a parameter to a procedure in the database.

SYSPROCS Consolidated View [page 1987]
The SYSPROCS view shows the procedure or function name, the name of its creator and any comments recorded for the procedure or function.

SYSPUBLICATIONS Consolidated View [page 1987]
Each row in the SYSPUBLICATIONS view describes a publication.

SYSREMOTEOPTION2 Consolidated View [page 1988]
Joins together, and presents in a more readable format, the columns from SYSREMOTEOPTION and SYSREMOTEOPTIONTYPE system views.

SYSREMOTEOPTIONS Consolidated View [page 1988]
Each row of the SYSREMOTEOPTIONS view describes the values of a message link parameter.

SYSREMOTETYPES Consolidated View [page 1989]
Each row of the SYSREMOTETYPES view describes one remote message type, including the publisher address.

SYSREMOTEUSERS Consolidated View [page 1989]
Each row of the SYSREMOTEUSERS view describes a user ID with the REMOTE system privilege (a subscriber), together with the status of messages that were sent to and from that user.

SYSROLEGRANTS Consolidated View [page 1990]
The SYSROLEGRANTS system view stores information about role membership and type of membership, just like the SYSROLEGRANT system view does. However, SYSROLEGRANTS includes role names and grantee names (not just IDs). The underlying system tables for this view are ISYSROLEGRANT and ISYSUSER.

SYSSUBSCRIPTIONS Consolidated View [page 1992]
Each row describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication.

SYSSYNC2 Consolidated View [page 1992]
The SYSSYNC2 view provides public access to the data found in the SYSSYNC system view (information related to synchronization) without exposing potentially sensitive data.

SYSSYNCPUBLICATIONDEFAULTS Consolidated View [page 1993]
The SYSSYNCPUBLICATIONDEFAULTS view provides the default synchronization settings associated with publications involved in synchronization.

SYSSYNCS Consolidated View [page 1993]
The SYSSYNCS view contains information relating to synchronization.

SYSSYNCSCRIPTS Consolidated View [page 1994]
Each row in the SYSSYNCSCRIPTS view identifies a stored procedure for scripted upload. This view is almost identical to the SYSSYNCSCRIPT system view, except that the values are in human-readable format, as opposed to raw data.

SYSSYNCSUBSCRIPTIONS Consolidated View [page 1994]
The SYSSYNCSUBSCRIPTIONS view contains the synchronization settings associated with synchronization subscriptions.

SYSSYNCUSERS Consolidated View [page 1995]
A view of synchronization settings associated with synchronization users.

SYSTABAUTH Consolidated View [page 1996]
The SYSTABAUTH view contains information from the SYSTABLEPERM system view, but in a more readable format.

SYSTRIGGERS Consolidated View [page 1996]
Each row in the SYSTRIGGERS view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE).

SYSUSEROPTIONS Consolidated View [page 1997]
The SYSUSEROPTIONS view contains the option settings that are in effect for each user. If a user has no setting for an option, this view displays the public setting for the option.

SYSVIEWS Consolidated View [page 1998]
Each row of the SYSVIEWS view describes one view, including its view definition.

1.7.2.1 ST_GEOMETRY_COLUMNS Consolidated View

Each row of the ST_GEOMETRY_COLUMNS system view describes a spatial column defined in the database.

Column name	Data type	Description
table_catalog	VARCHAR(128)	For internal use.

Column name	Data type	Description
table_schema	CHAR(128)	The name of the schema to which the table containing the spatial column belongs. This is equivalent to the table owner.
table_name	CHAR(128)	The name of the table containing the spatial column.
column_name	CHAR(128)	The name of the spatial column.
srs_name	CHAR(128)	The name of the SRS that is associated with the spatial column. If an SRS is not associated with the column, then srs_name is NULL.
srs_id	INTEGER	The SRID for the SRS associated with the spatial column.
table_id	UNSIGNED INT	The numeric identifier for the table containing the column.
column_id	UNSIGNED INT	The numeric identifier for the column.
geometry_type_name	VARCHAR(32767)	The spatial data type of the geometries contained in the column (for example, ST_Point, ST_Geometry, and so on.

Related Information

Spatial Reference Systems (SRS) and Spatial Reference Identifiers (SRID)
Supported Spatial Data Types and Their Hierarchy

1.7.2.2 ST_SPATIAL_REFERENCE_SYSTEMS Consolidated View

Each row of the ST_SPATIAL_REFERENCE_SYSTEMS system view describes an SRS defined in the database. This view offers a slightly different amount of information than the SYSSPATIALREFERENCESYSTEM system view.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	For system use only.
owner	UNSIGNED INT	The owner of the SRS.
srs_name	CHAR(128)	The name of the SRS.
srs_id	INTEGER	The numeric identifier (SRID) for the spatial reference system.

Column name	Data type	Description
srs_type	CHAR(11)	The type of SRS as defined by the SQL/MM standard. Values can be one of: **GEOGRAPHIC** This is for SRSs based on georeferenced coordinate systems with axes of latitude, longitude (and elevation). These SRSs are of type PLANAR or ROUND EARTH. **PROJECTED** This is for SRSs based on georeferenced coordinate systems that do not have axes of latitude and longitude. These SRSs are of type PLANAR. **ENGINEERING** This is for SRSs based on non-georeferenced coordinate systems. These SRSs are of type PLANAR. **GEOCENTRIC** Unsupported. **COMPOUND** Unsupported. **VERTICAL** Unsupported. If srs_type is empty, the type is unspecified.
round_earth	CHAR(1)	Whether the SRS type is ROUND EARTH (Y) or PLANAR (N).
axis_order	CHAR(12)	Describes how the database server interprets points with regards to latitude and longitude (for example when using the ST_Lat and ST_Long methods). For non-geographic spatial reference systems, the axis order is x/y/z/m. For geographic spatial reference systems, the default axis order is long/lat/z/m; lat/long/z/m is also supported.
snap_to_grid	DOUBLE	Defines the size of the grid the database server uses when performing calculations.
tolerance	DOUBLE	Defines the precision to use when comparing points.

Column name	Data type	Description
semi_major_axis	DOUBLE	Distance from center of the ellipsoid to the equator for a ROUND EARTH SRS.
semi_minor_axis	DOUBLE	Distance from center of the ellipsoid to the poles for a ROUND EARTH SRS.
inv_flattening	DOUBLE	The inverse flattening used for the ellipsoid in a ROUND EARTH SRS. This is a ratio created by the following equation: `1/f = (semi-major-axis) / (semi-major-axis - semi-minor-axis)`
min_x	DOUBLE	The minimum x value allowed in coordinates.
max_x	DOUBLE	The maximum x value allowed in coordinates.
min_y	DOUBLE	The minimum y value allowed in coordinates.
max_y	DOUBLE	The maximum y value allowed in coordinates.
min_z	DOUBLE	The minimum z value allowed in coordinates.
max_z	DOUBLE	The maximum z value allowed in coordinates.
min_m	DOUBLE	The minimum m value allowed in coordinates.
max_m	DOUBLE	The maximum m value allowed in coordinates.
min_lat	DOUBLE	The minimum latitude value allowed for coordinates.
max_lat	DOUBLE	The maximum latitude value allowed for coordinates.
min_long	DOUBLE	The minimum longitude value allowed in coordinates.
max_long	DOUBLE	The maximum longitude value allowed in coordinates.
organization	LONG VARCHAR	The name of the organization that created the coordinate system used by the spatial reference system.
organization_coordsys_id	INTEGER	The ID given to the coordinate system by the organization that created it.
linear_unit_of_measure	CHAR(128)	The linear unit of measurement used by the SRS.

Column name	Data type	Description
angular_unit_of_measure	CHAR(128)	The angular unit of measurement used by the SRS.
polygon_format	LONG VARCHAR	The orientation of the rings in a polygon. One of CounterClockwise, ClockWise, or EvenOdd.
storage_format	LONG VARCHAR	Whether the data is stored in normalized format (Internal), unnormalized format (Original), or both (Mixed).
definition	LONG VARCHAR	Additional definition settings.
transform_definition	LONG VARCHAR	Transform definition settings for use when transforming data from this SRS to another.
description	LONG VARCHAR	Description of the SRS.

1.7.2.3 ST_UNITS_OF_MEASURE Consolidated View

Each row of the ST_UNITS_OF_MEASURE system view describes a unit of measure defined in the database. This view offers more information than the SYSUNITOFMEASURE system view.

Column name	Data type	Description
object_id	UNSIGNED BIGINT	For system use only.
owner	UNSIGNED INT	The owner of the unit of measure.
unit_name	CHAR(128)	The name of the unit of measure.
unit_type	CHAR(7)	Angular or linear.
conversion_factor	DOUBLE	The conversion factor for the unit of measure.
description	LONG VARCHAR	Description for the unit of measure.

1.7.2.4 SYSARTICLECOLS Consolidated View

Each row in the SYSARTICLECOLS view identifies a column in an article.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSARTICLECOLS"
  as select p.publication_name,t.table_name,c.column_name
    from SYS.ISYSARTICLECOL as ac
      join SYS.ISYSPUBLICATION as p on p.publication_id = ac.publication_id
      join SYS.ISYSTAB as t on t.table_id = ac.table_id
      join SYS.ISYSTABCOL as c on c.table_id = ac.table_id
      and c.column_id = ac.column_id
```

Related Information

SYSARTICLECOL System View [page 1899]
SYSPUBLICATION System View [page 1935]
SYSTAB System View [page 1950]
SYSTABCOL System View [page 1953]

1.7.2.5 SYSARTICLES Consolidated View

Each row in the SYSARTICLES view describes an article in a publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSARTICLES"
  as select u1.user_name as publication_owner,p.publication_name,
    u2.user_name as table_owner,t.table_name,
    a.where_expr,a.subscribe_by_expr,a.alias
    from SYS.ISYSARTICLE as a
      join SYS.ISYSPUBLICATION as p on(a.publication_id = p.publication_id)
      join SYS.ISYSTAB as t on(a.table_id = t.table_id)
      join SYS.ISYSUSER as u1 on(p.creator = u1.user_id)
      join SYS.ISYSUSER as u2 on(t.creator = u2.user_id)
```

Related Information

SYSARTICLE System View [page 1898]
SYSPUBLICATION System View [page 1935]
SYSTAB System View [page 1950]
SYSUSER System View [page 1965]

1.7.2.6 SYSCAPABILITIES Consolidated View

Each row in the SYSCAPABILITIES view specifies the status of a capability for a remote database server. This view gets its data from the ISYSCAPABILITY system table.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCAPABILITIES"
  as select
ISYSCAPABILITY.capid,ISYSCAPABILITY.srvid,property('RemoteCapability',ISYSCAPABIL
ITY.capid) as capname,ISYSCAPABILITY.capvalue
    from SYS.ISYSCAPABILITY
```

Related Information

SYSCAPABILITY System View [page 1899]
SYSCAPABILITYNAME System View [page 1900]

1.7.2.7 SYSCATALOG Consolidated View

Each row in the SYSCATALOG view describes a system table.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCATALOG"( creator,
  tname,dbspacename,tabletype,ncols,primary_key,"check",
  remarks )
  as select u.user_name,tab.table_name,dbs.dbspace_name,
    if tab.table_type_str = 'BASE' then 'TABLE' else tab.table_type_str endif,
    (select count() from SYS.ISYSTABCOL
      where ISYSTABCOL.table_id = tab.table_id),
    if ix.index_id is null then 'N' else 'Y' endif,
    null,
    rmk.remarks
    from SYS.SYSTAB as tab
      join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.dbspace_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
      left outer join SYS.ISYSIDX as ix on(tab.table_id = ix.table_id and
ix.index_id = 0)
      left outer join SYS.ISYSREMARK as rmk on(tab.object_id = rmk.object_id)
```

Related Information

SYSTAB System View [page 1950]
SYSTABCOL System View [page 1953]
SYSDBSPACE System View [page 1904]
SYSUSER System View [page 1965]
SYSIDX System View [page 1916]
SYSREMARK System View [page 1936]

1.7.2.8 SYSCOLAUTH Consolidated View

Each row in the SYSCOLAUTH view describes the set of privileges (UPDATE, SELECT, or REFERENCES) granted on a column.

The SYSCOLAUTH view provides a user-friendly presentation of data in the SYSCOLPERM system view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLAUTH"( grantor,grantee,creator,tname,colname,
  privilege_type,is_grantable )
  as select u1.user_name,u2.user_name,u3.user_name,tab.table_name,
    col.column_name,cp.privilege_type,cp.is_grantable
    from SYS.ISYSCOLPERM as cp
      join SYS.ISYSUSER as u1 on u1.user_id = cp.grantor
      join SYS.ISYSUSER as u2 on u2.user_id = cp.grantee
      join SYS.ISYSTAB as tab on tab.table_id = cp.table_id
      join SYS.ISYSUSER as u3 on u3.user_id = tab.creator
      join SYS.ISYSTABCOL as col on col.table_id = cp.table_id
      and col.column_id = cp.column_id
```

Related Information

SYSCOLPERM System View [page 1901]
SYSTABCOL System View [page 1953]
SYSUSER System View [page 1965]
SYSTAB System View [page 1950]

1.7.2.9 SYSCOLSTATS Consolidated View

The SYSCOLSTATS view contains the column statistics that are stored as histograms and used by the optimizer.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLSTATS" AS SELECT u.user_name, t.table_name,
c.column_name, s.format_id,
   dateadd(mi, PROPERTY('TimeZoneAdjustment'), s.update_time) as update_time,
s.density, s.max_steps, s.actual_steps,
   s.step_values, s.frequencies , TODATETIMEOFFSET( s.update_time, 0 ) as
update_time_utc
FROM SYS.ISYSCOLSTAT s
JOIN SYS.ISYSTABCOL c on (s.table_id = c.table_id and s.column_id =
c.column_id)
JOIN SYS.ISYSTAB t on (t.table_id = c.table_id)
JOIN SYS.ISYSUSER u on (u.user_id = t.creator)
```

Related Information

SYSCOLSTAT System View [page 1902]
SYSTABCOL System View [page 1953]
SYSTAB System View [page 1950]
SYSUSER System View [page 1965]

1.7.2.10 SYSCOLUMNS Consolidated View

Each row in the SYSCOLUMNS view describes one column of each table and view in the catalog.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLUMNS"( creator,cname,tname,coltype,nulls,length,
  syslength,in_primary_key,colno,default_value,
  column_kind,remarks )
  as select u.user_name,col.column_name,tab.table_name,dom.domain_name,
    col.nulls,col.width,col.scale,if ixcol.sequence is null then 'N' else 'Y'
endif,col.column_id,
    col."default",col.column_type,rmk.remarks
    from SYS.SYSTABCOL as col
      left outer join SYS.ISYSIDXCOL as ixcol on(col.table_id = ixcol.table_id
and col.column_id = ixcol.column_id and ixcol.index_id = 0)
      join SYS.ISYSTAB as tab on(tab.table_id = col.table_id)
      join SYS.ISYSDOMAIN as dom on(dom.domain_id = col.domain_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
      left outer join SYS.ISYSREMARK as rmk on(col.object_id = rmk.object_id)
```

Related Information

SYSTABCOL System View [page 1953]
SYSIDXCOL System View [page 1918]
SYSTAB System View [page 1950]
SYSDOMAIN System View [page 1905]
SYSUSER System View [page 1965]
SYSREMARK System View [page 1936]

1.7.2.11 SYSFOREIGNKEYS Consolidated View

Each row in the SYSFOREIGNKEYS view describes one foreign key for each table in the catalog.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFOREIGNKEYS"( foreign_creator,
  foreign_tname,
  primary_creator,primary_tname,role,columns )
  as select fk_up.user_name,fk_tab.table_name,pk_up.user_name,
    pk_tab.table_name,ix.index_name,
    (select list(string(fk_col.column_name,' IS ',
      pk_col.column_name)
      order by fkc.table_id,fkc.index_id,fkc."sequence")
      from SYS.ISYSIDXCOL as fkc
        join SYS.ISYSTABCOL as fk_col on(
        fkc.table_id = fk_col.table_id
        and fkc.column_id = fk_col.column_id)
        ,SYS.ISYSTABCOL as pk_col
      where fkc.table_id = fk.foreign_table_id
```

```
      and fkc.index_id = fk.foreign_index_id
      and pk_col.table_id = fk.primary_table_id
      and pk_col.column_id = fkc.primary_column_id)
    from SYS.ISYSFKEY as fk
      join SYS.ISYSTAB as fk_tab on fk_tab.table_id = fk.foreign_table_id
      join SYS.ISYSUSER as fk_up on fk_up.user_id = fk_tab.creator
      join SYS.ISYSTAB as pk_tab on pk_tab.table_id = fk.primary_table_id
      join SYS.ISYSUSER as pk_up on pk_up.user_id = pk_tab.creator
      join SYS.ISYSIDX as ix on ix.table_id = fk.foreign_table_id and
ix.index_id = fk.foreign_index_id
```

Related Information

SYSTAB System View [page 1950]
SYSTABCOL System View [page 1953]
SYSIDX System View [page 1916]
SYSIDXCOL System View [page 1918]
SYSFKEY System View [page 1912]
SYSUSER System View [page 1965]
SYSDOMAIN System View [page 1905]
SYSREMARK System View [page 1936]

1.7.2.12 SYSINDEXES Consolidated View

Each row in the SYSINDEXES view describes one index in the database. As an alternative to this view, you could also use the SYSIDX and SYSIDXCOL system views.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSINDEXES"( icreator,
  iname,fname,creator,tname,indextype,
  colnames,interval,level_num )
  as select u.user_name,idx.index_name,dbs.dbspace_name,u.user_name,
    tab.table_name,
    case idx.index_category
    when 1 then 'Primary Key'
    when 2 then 'Foreign Key'
    when 3 then(
      if idx."unique" = 4 then 'Non-unique'
      else if idx."unique" = 2 then 'UNIQUE constraint'
        else if idx."unique" = 5 then 'UNIQUE NULLS NOT DISTINCT'
          else 'UNIQUE'
          endif
        endif
      endif) when 4 then 'Text Index' end,(select list(string(c.column_name,
      if ixc."order" = 'A' then ' ASC' else ' DESC' endif) order by
      ixc.table_id asc,ixc.index_id asc,ixc.sequence asc)
      from SYS.ISYSIDXCOL as ixc
        join SYS.ISYSTABCOL as c on(
        c.table_id = ixc.table_id
        and c.column_id = ixc.column_id)
      where ixc.index_id = idx.index_id
```

```
      and ixc.table_id = idx.table_id),
    0,0
    from SYS.ISYSTAB as tab
      join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.dbspace_id)
      join SYS.ISYSIDX as idx on(idx.table_id = tab.table_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
```

Related Information

SYSIDX System View [page 1916]
SYSTABCOL System View [page 1953]
SYSTAB System View [page 1950]
SYSDBSPACE System View [page 1904]
SYSIDXCOL System View [page 1918]
SYSUSER System View [page 1965]

1.7.2.13 SYSOPTIONS Consolidated View

Each row in the SYSOPTIONS view describes one option created using the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user define the default settings to be used for users that do not have their own setting.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSOPTIONS"( user_name,"option",setting )
  as select u.user_name,opt."option",opt.setting
    from SYS.ISYSOPTION as opt
      join SYS.ISYSUSER as u on opt.user_id = u.user_id
```

Related Information

SYSOPTION System View [page 1929]
SYSUSER System View [page 1965]

1.7.2.14 SYSPROCAUTH Consolidated View

Each row in the SYSPROCAUTH view describes a set of privileges granted on a procedure. As an alternative, you can also use the SYSPROCPERM system view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSPROCAUTH"( grantee,
  creator,procname )
  as select u1.user_name,u2.user_name,p.proc_name
    from SYS.ISYSPROCEDURE as p
      join SYS.ISYSPROCPERM as pp on(p.proc_id = pp.proc_id)
      join SYS.ISYSUSER as u1 on u1.user_id = pp.grantee
      join SYS.ISYSUSER as u2 on u2.user_id = p.creator
```

Related Information

SYSPROCEDURE System View [page 1931]
SYSPROCPERM System View [page 1934]
SYSUSER System View [page 1965]

1.7.2.15 SYSPROCPARMS Consolidated View

Each row in the SYSPROCPARMS view describes a parameter to a procedure in the database.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSPROCPARMS"( creator,
  procname,parmname,parm_id,parmtype,parmmode,parmdomain,
  length,scale,"default",user_type )
  as select up.user_name,p.proc_name,pp.parm_name,pp.parm_id,pp.parm_type,
    if pp.parm_mode_in = 'Y' and pp.parm_mode_out = 'N' then 'IN'
    else if pp.parm_mode_in = 'N' and pp.parm_mode_out = 'Y' then 'OUT'
      else 'INOUT'
      endif
    endif,dom.domain_name,pp.width,pp.scale,pp."default",ut.type_name
    from SYS.SYSPROCPARM as pp
      join SYS.ISYSPROCEDURE as p on p.proc_id = pp.proc_id
      join SYS.ISYSUSER as up on up.user_id = p.creator
      join SYS.ISYSDOMAIN as dom on dom.domain_id = pp.domain_id
      left outer join SYS.ISYSUSERTYPE as ut on ut.type_id = pp.user_type
```

Related Information

SYSPROCPARM System View [page 1932]
SYSPROCEDURE System View [page 1931]

SYSUSER System View [page 1965]
SYSDOMAIN System View [page 1905]
SYSUSERTYPE System View [page 1968]

1.7.2.16 SYSPROCS Consolidated View

The SYSPROCS view shows the procedure or function name, the name of its creator and any comments recorded for the procedure or function.

The tables and columns that make up this view are provided in the ALTER VIEW statement below.

```
ALTER VIEW "SYS"."SYSPROCS"( creator,
  procname,remarks )
  as select u.user_name,p.proc_name,r.remarks
    from SYS.ISYSPROCEDURE as p
      join SYS.ISYSUSER as u on u.user_id = p.creator
      left outer join SYS.ISYSREMARK as r on(p.object_id = r.object_id)
```

Related Information

SYSPROCEDURE System View [page 1931]
SYSUSER System View [page 1965]
SYSREMARK System View [page 1936]

1.7.2.17 SYSPUBLICATIONS Consolidated View

Each row in the SYSPUBLICATIONS view describes a publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSPUBLICATIONS"
  as select u.user_name as creator,
    p.publication_name,
    r.remarks,
    p.type,
    case p.sync_type
    when 0 then 'logscan'
    when 1 then 'scripted upload'
    when 2 then 'download only'
    else 'invalid'
    end as sync_type
    from SYS.ISYSPUBLICATION as p
      join SYS.ISYSUSER as u on u.user_id = p.creator
      left outer join SYS.ISYSREMARK as r on(p.object_id = r.object_id)
```

Related Information

SYSPUBLICATION System View [page 1935]
SYSREMARK System View [page 1936]

1.7.2.18 SYSREMOTEOPTION2 Consolidated View

Joins together, and presents in a more readable format, the columns from SYSREMOTEOPTION and SYSREMOTEOPTIONTYPE system views.

Values in the setting column are hidden from users that do not have the SELECT ANY TABLE system privilege.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTEOPTION2"
  as select ISYSREMOTEOPTION.option_id,
    ISYSREMOTEOPTION.user_id,
    SYS.HIDE_FROM_NON_DBA(ISYSREMOTEOPTION.setting) as setting
    from SYS.ISYSREMOTEOPTION
```

Related Information

SYSREMOTEOPTION System View [page 1936]

1.7.2.19 SYSREMOTEOPTIONS Consolidated View

Each row of the SYSREMOTEOPTIONS view describes the values of a message link parameter.

Values in the setting column are hidden from users that do not have the SELECT ANY TABLE system privilege. The SYSREMOTEOPTION2 view provides public access to the insensitive data.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTEOPTIONS"
  as select srt.type_name,
    sup.user_name,
    srot."option",
    SYS.HIDE_FROM_NON_DBA(sro.setting) as setting
    from SYS.ISYSREMOTETYPE as srt
      ,SYS.ISYSREMOTEOPTIONTYPE as srot
      ,SYS.ISYSREMOTEOPTION as sro
      ,SYS.ISYSUSER as sup
    where srt.type_id = srot.type_id
    and srot.option_id = sro.option_id
    and sro.user_id = sup.user_id
```

Related Information

SYSREMOTETYPE System View [page 1937]
SYSREMOTEOPTIONTYPE System View [page 1936]
SYSREMOTEOPTION System View [page 1936]
SYSUSER System View [page 1965]

1.7.2.20 SYSREMOTETYPES Consolidated View

Each row of the SYSREMOTETYPES view describes one remote message type, including the publisher address.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTETYPES"
  as select rt.type_id,rt.type_name,rt.publisher_address,rm.remarks
    from SYS.ISYSREMOTETYPE as rt
      left outer join SYS.ISYSREMARK as rm on(rt.object_id = rm.object_id)
```

Related Information

SYSREMOTETYPE System View [page 1937]
SYSREMARK System View [page 1936]

1.7.2.21 SYSREMOTEUSERS Consolidated View

Each row of the SYSREMOTEUSERS view describes a user ID with the REMOTE system privilege (a subscriber), together with the status of messages that were sent to and from that user.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTEUSERS" AS SELECT u.user_name, r.consolidate,
t.type_name, r.address, r.frequency, r.send_time,
(if r.frequency = 'A' then NULL
else if r.frequency = 'P' then
if r.time_sent IS NULL then CURRENT TIMESTAMP
else (select min( minutes( dateadd(mi, PROPERTY('TimeZoneAdjustment'),
a.time_sent),
60*hour(a.send_time) + minute( seconds( a.send_time, 59 ) ) ) )
FROM SYS.ISYSREMOTEUSER a WHERE a.frequency = 'P' AND a.send_time =
r.send_time ) endif
else if CURRENT DATE + r.send_time > coalesce( dateadd(mi,
PROPERTY('TimeZoneAdjustment'), r.time_sent), CURRENT TIMESTAMP)
then CURRENT DATE + r.send_time
else CURRENT DATE + r.send_time + 1
endif   endif  endif) as next_send, r.log_send ,
```

```
dateadd(mi, PROPERTY('TimeZoneAdjustment'), r.time_sent)
as time_sent , r.log_sent, r.confirm_sent, r.send_count, r.resend_count,
dateadd(mi, PROPERTY('TimeZoneAdjustment'), r.time_received) as time_received ,
r.log_received, r.confirm_received, r.receive_count, r.rereceive_count ,
TODATETIMEOFFSET( r.time_sent, 0 ) as time_sent_utc ,
TODATETIMEOFFSET( r.time_received, 0 ) as time_received_utc
FROM SYS.ISYSREMOTEUSER r JOIN SYS.ISYSUSER u ON ( u.user_id = r.user_id ) JOIN
SYS.ISYSREMOTETYPE t ON ( t.type_id = r.type_id )
```

Related Information

SYSREMOTEUSER System View [page 1937]
SYSUSER System View [page 1965]
SYSREMOTETYPE System View [page 1937]

1.7.2.22 SYSROLEGRANTS Consolidated View

The SYSROLEGRANTS system view stores information about role membership and type of membership, just like the SYSROLEGRANT system view does. However, SYSROLEGRANTS includes role names and grantee names (not just IDs). The underlying system tables for this view are ISYSROLEGRANT and ISYSUSER.

Column name	Data type	Description
grant_id	UNSIGNED INT	ID used to identify each GRANT statement.
role_id	UNSIGNED INT	ID of the role being granted, as per ISYSUSER.
role_name	CHAR(128)	The name of the role.
grantee	UNISIGNED INT	ID of the user being granted the role, as per ISYSUSER.
grantee_name	CHAR(128)	The name of the grantee.

Column name	Data type	Description
grant_type	TINYINT	Describes type of grant using 3 bits. The first bit from the right is whether privilege has been granted. The second digit is whether administration rights have been given. The third digit is whether system privileges are inheritable. **001** Privilege granted, with no inheritance, and no administration rights. Applicable only for legacy non-inheritable authorities except DBA and REMOTE DBA. **101** Privilege granted, with inheritance, but no administration rights. **110** Only administration rights have been granted. **111** Privilege granted, with inheritance, and with administration rights.
grant_scope	TINYINT	Used by SET USER and CHANGE PASSWORD to set the scope of the grant. Values can be one or more of the following: **1** ANY **2** User list **4** Role list
grantor	CHAR(128)	The name of the grantor.

1.7.2.23 SYSSUBSCRIPTIONS Consolidated View

Each row describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSSUBSCRIPTIONS"
  as select p.publication_name,u.user_name,s.subscribe_by,s.created,
    s.started
    from SYS.ISYSSUBSCRIPTION as s
      join SYS.ISYSPUBLICATION as p on(p.publication_id = s.publication_id)
      join SYS.ISYSUSER as u on u.user_id = s.user_id
```

Related Information

SYSSUBSCRIPTION System View [page 1947]
SYSPUBLICATION System View [page 1935]
SYSUSER System View [page 1965]

1.7.2.24 SYSSYNC2 Consolidated View

The SYSSYNC2 view provides public access to the data found in the SYSSYNC system view (information related to synchronization) without exposing potentially sensitive data.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular column, use the links provided beneath the view definition.

The server_connect and option columns display three asterisks (***) if a value is present in the database and NULL if no value is present.

```
ALTER VIEW "SYS"."SYSSYNC2"
  as select "ISYSSYNC"."sync_id",
    "ISYSSYNC"."type",
    "ISYSSYNC"."publication_id",
    "ISYSSYNC"."progress",
    "ISYSSYNC"."site_name",
    if "ISYSSYNC"."option" is null then null else '***' endif as "option",
    if "ISYSSYNC"."server_connect" is null then null else '***' endif as
"server_connect",
    "ISYSSYNC"."server_conn_type",
    "ISYSSYNC"."last_download_time",
    "ISYSSYNC"."last_upload_time",
    "ISYSSYNC"."created",
    "ISYSSYNC"."log_sent",
    "ISYSSYNC"."generation_number",
    "ISYSSYNC"."extended_state",
    "ISYSSYNC"."script_version",
    "ISYSSYNC"."subscription_name",
    "ISYSSYNC"."server_protocol"
    from "SYS"."ISYSSYNC"
```

Related Information

SYSSYNC System View [page 1948]

1.7.2.25 SYSSYNCPUBLICATIONDEFAULTS Consolidated View

The SYSSYNCPUBLICATIONDEFAULTS view provides the default synchronization settings associated with publications involved in synchronization.

The tables and columns that make up this view are provided in the SQL statement below.

The server_connect and option columns display three asterisks (***) if a value is present in the database and NULL if no value is present.

```
ALTER VIEW "SYS"."SYSSYNCPUBLICATIONDEFAULTS"
  as select "s"."sync_id",
    "p"."publication_name",
    "s"."option",
    "s"."server_connect",
    "s"."server_conn_type"
    from "SYS"."SYSSYNC2" as "s" join "SYS"."SYSPUBLICATION" as "p"
on("p"."publication_id" = "s"."publication_id") where
    "s"."site_name" is null
```

Related Information

SYSSYNC2 Consolidated View [page 1992]
SYSPUBLICATION System View [page 1935]

1.7.2.26 SYSSYNCS Consolidated View

The SYSSYNCS view contains information relating to synchronization.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

The server_connect and option columns display three asterisks (***) if a value is present in the database and NULL if no value is present.

The underlying view for this consolidated view is SYSSYNC2.

```
ALTER VIEW "SYS"."SYSSYNCS"
  as select "p"."publication_name","s"."progress","s"."site_name",
    "s"."option",
    "s"."server_connect",
    "s"."server_conn_type","s"."last_download_time",
    "s"."last_upload_time","s"."created","s"."log_sent","s"."generation_number",
    "s"."extended_state"
```

```
    from "SYS"."SYSSYNC2" as "s"
      left outer join "SYS"."SYSPUBLICATION" as "p"
      on "p"."publication_id" = "s"."publication_id"
```

Related Information

SYSSYNC2 Consolidated View [page 1992]
SYSPUBLICATION System View [page 1935]

1.7.2.27 SYSSYNCSCRIPTS Consolidated View

Each row in the SYSSYNCSCRIPTS view identifies a stored procedure for scripted upload. This view is almost identical to the SYSSYNCSCRIPT system view, except that the values are in human-readable format, as opposed to raw data.

```
ALTER VIEW "SYS"."SYSSYNCSCRIPTS"
  as select p.publication_name,
    t.table_name,
    case s.type
    when 0 then 'upload insert'
    when 1 then 'upload delete'
    when 2 then 'upload update'
    else 'unknown'
    end as type,
    c.proc_name
    from SYS.ISYSSYNCSCRIPT as s
      join SYS.ISYSPUBLICATION as p on p.object_id = s.pub_object_id
      join SYS.ISYSTAB as t on t.object_id = s.table_object_id
      join SYS.ISYSPROCEDURE as c on c.object_id = s.proc_object_id
```

Related Information

Scripted Upload
SYSSYNCSCRIPT System View [page 1950]
SYSPUBLICATION System View [page 1935]
SYSTAB System View [page 1950]
SYSPROCEDURE System View [page 1931]

1.7.2.28 SYSSYNCSUBSCRIPTIONS Consolidated View

The SYSSYNCSUBSCRIPTIONS view contains the synchronization settings associated with synchronization subscriptions.

The tables and columns that make up this view are provided in the SQL statement below.

The server_connect and option columns display three asterisks (***) if a value is present in the database and NULL if no value is present.

```
ALTER VIEW "SYS"."SYSSYNCSUBSCRIPTIONS"
  as select "s"."sync_id",
    "p"."publication_name",
    "s"."progress",
    "s"."site_name",
    "s"."option",
    "s"."server_connect",
    "s"."server_conn_type",
    "s"."last_download_time",
    "s"."last_upload_time",
    "s"."created",
    "s"."log_sent",
    "s"."generation_number",
    "s"."extended_state"
    from "SYS"."SYSSYNC2" as "s" join "SYS"."SYSPUBLICATION" as "p"
on("p"."publication_id" = "s"."publication_id")
    where "s"."publication_id" is not null and
    "s"."site_name" is not null and exists
    (select 1 from "SYS"."SYSSYNCUSERS" as "u"
      where "s"."site_name" = "u"."site_name")
```

Related Information

SYSSYNC2 Consolidated View [page 1992]
SYSPUBLICATION System View [page 1935]
SYSSYNCUSERS Consolidated View [page 1995]

1.7.2.29 SYSSYNCUSERS Consolidated View

A view of synchronization settings associated with synchronization users.

The tables and columns that make up this view are provided in the SQL statement below.

The server_connect and option columns display three asterisks (***) if a value is present in the database and NULL if no value is present.

```
ALTER VIEW "SYS"."SYSSYNCUSERS"
  as select "SYSSYNC2"."sync_id",
    "SYSSYNC2"."site_name",
    "SYSSYNC2"."option",
    "SYSSYNC2"."server_connect",
    "SYSSYNC2"."server_conn_type"
    from "SYS"."SYSSYNC2" where
    "SYSSYNC2"."publication_id" is null
```

Related Information

SYSSYNC2 Consolidated View [page 1992]

1.7.2.30 SYSTABAUTH Consolidated View

The SYSTABAUTH view contains information from the SYSTABLEPERM system view, but in a more readable format.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSTABAUTH"( grantor,
  grantee,screator,stname,tcreator,ttname,
  selectauth,insertauth,deleteauth,
  updateauth,updatecols,alterauth,referenceauth,
  loadauth,truncateauth )
  as select u1.user_name,u2.user_name,u3.user_name,tab1.table_name,
    u4.user_name,tab2.table_name,tp.selectauth,tp.insertauth,
    tp.deleteauth,tp.updateauth,tp.updatecols,tp.alterauth,
    tp.referenceauth,tp.loadauth,tp.truncateauth
    from SYS.ISYSTABLEPERM as tp
      join SYS.ISYSUSER as u1 on u1.user_id = tp.grantor
      join SYS.ISYSUSER as u2 on u2.user_id = tp.grantee
      join SYS.ISYSTAB as tab1 on tab1.table_id = tp.stable_id
      join SYS.ISYSUSER as u3 on u3.user_id = tab1.creator
      join SYS.ISYSTAB as tab2 on tab2.table_id = tp.stable_id
      join SYS.ISYSUSER as u4 on u4.user_id = tab2.creator
```

Related Information

SYSTABLEPERM System View [page 1956]
SYSUSER System View [page 1965]
SYSTAB System View [page 1950]

1.7.2.31 SYSTRIGGERS Consolidated View

Each row in the SYSTRIGGERS view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE).

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSTRIGGERS"( owner,
  trigname,tname,event,trigtime,trigdefn )
  as select u.user_name,trig.trigger_name,tab.table_name,
    if trig.event = 'I' then 'INSERT'
    else if trig.event = 'U' then 'UPDATE'
      else if trig.event = 'C' then 'UPDATE'
        else if trig.event = 'D' then 'DELETE'
          else if trig.event = 'A' then 'INSERT,DELETE'
            else if trig.event = 'B' then 'INSERT,UPDATE'
              else if trig.event = 'E' then 'DELETE,UPDATE'
                else 'INSERT,DELETE,UPDATE'
                endif
              endif
```

```
            endif
          endif
        endif
      endif
    endif,if trig.trigger_time = 'B' or trig.trigger_time = 'P' then 'BEFORE'
    else if trig.trigger_time = 'A' or trig.trigger_time = 'S' then 'AFTER'
      else if trig.trigger_time = 'R' then 'RESOLVE'
        else 'INSTEAD OF'
        endif
      endif
    endif,trig.trigger_defn
    from SYS.ISYSTRIGGER as trig
      join SYS.ISYSTAB as tab on(tab.table_id = trig.table_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator where
    trig.foreign_table_id is null
```

Related Information

SYSTRIGGER System View [page 1962]
SYSTAB System View [page 1950]
SYSUSER System View [page 1965]

1.7.2.32 SYSUSEROPTIONS Consolidated View

The SYSUSEROPTIONS view contains the option settings that are in effect for each user. If a user has no setting for an option, this view displays the public setting for the option.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSEROPTIONS"( user_name,
  "option",setting )
  as select u.user_name,
    o."option",
    isnull((select s.setting
      from SYS.ISYSOPTION as s
      where s.user_id = u.user_id
      and s."option" = o."option"),
    o.setting)
    from SYS.SYSOPTIONS as o,SYS.ISYSUSER as u
    where o.user_name = 'PUBLIC'
```

Related Information

SYSOPTIONS Consolidated View [page 1985]
SYSUSER System View [page 1965]

1.7.2.33 SYSVIEWS Consolidated View

Each row of the SYSVIEWS view describes one view, including its view definition.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSVIEWS"( vcreator,
  viewname,viewtext )
  as select u.user_name,t.table_name,v.view_def
    from SYS.ISYSTAB as t
      join SYS.ISYSVIEW as v on(t.object_id = v.view_object_id)
      join SYS.ISYSUSER as u on(u.user_id = t.creator)
```

Related Information

SYSTAB System View [page 1950]
SYSVIEW System View [page 1970]
SYSUSER System View [page 1965]

1.7.3 Compatibility Views

Compatibility views are views that are provided for compatibility with versions of the software that are 10 and earlier. Where possible use system and consolidated views instead, as support may diminish for some compatibility views in future releases.

In this section:

SYSCOLLATION Compatibility View (Deprecated) [page 2000]
The SYSCOLLATION compatibility view contains the collation sequence information for the database. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

SYSCOLLATIONMAPPINGS Compatibility View (Deprecated) [page 2000]
The SYSCOLLATIONMAPPINGS compatibility view contains only one row with the database collation mapping. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

SYSCOLUMN Compatibility View (Deprecated) [page 2001]
The SYSCOLUMN view is provided for compatibility with older versions of the software that offered a SYSCOLUMN system table.

SYSFILE Compatibility View (Deprecated) [page 2001]
Each row in the SYSFILE system view describes a dbspace for a database. Every database consists of one or more dbspaces; each dbspace corresponds to an operating system file.

SYSFKCOL Compatibility View (Deprecated) [page 2002]
Each row of SYSFKCOL describes the association between a foreign column in the foreign table of a relationship and the primary column in the primary table. This view is deprecated; use the SYSIDX and SYSIDXCOL system views instead.

SYSFOREIGNKEY Compatibility View (Deprecated) [page 2003]
The SYSFOREIGNKEY view is provided for compatibility with older versions of the software that offered a SYSFOREIGNKEY system table. However, the previous SYSFOREIGNKEY system table has been replaced by the ISYSFKEY system table, and its corresponding SYSFKEY system view, which you should use instead.

SYSGROUP Compatibility View [page 2003]
There is one row in the SYSGROUP system view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

SYSGROUPS Compatibility View [page 2004]
There is one row in the SYSGROUPS view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

SYSINDEX Compatibility View (Deprecated) [page 2004]
The SYSINDEX view is provided for compatibility with older versions of the software that offered a SYSINDEX system table. However, the SYSINDEX system table has been replaced by the ISYSIDX system table, and its corresponding SYSIDX system view, which you should use instead.

SYSINFO Compatibility View (Deprecated) [page 2005]
The SYSINFO view indicates the database characteristics, as defined when the database was created. It always contains only one row. This view is obtainable via built-in functions and is not kept in the catalog. Following is the definition for the SYSINFO view:

SYSIXCOL Compatibility View (Deprecated) [page 2006]
Each row of the SYSIXCOL describes a column in an index, and is provided for compatibility with older versions of the software that offered a SYSIXCOL system table.

SYSTABLE Compatibility View (Deprecated) [page 2006]
The SYSTABLE view is provided for compatibility with older versions of the software that offered a SYSTABLE system table. However, the SYSTABLE system table has been replaced by the ISYSTAB system table, and its corresponding SYSTAB system view, which you should use instead.

SYSUSERAUTH Compatibility View (Deprecated) [page 2007]
Each row of the SYSUSERAUTH view describes a user, without exposing their user ID and password hash. Instead, each user is identified by their user name.

SYSUSERAUTHORITY Compatibility View (Deprecated) [page 2008]
The SYSUSERAUTHORITY view is provided for compatibility with older versions of the software. Use the SYSROLEGRANTS consolidated view instead.

SYSUSERLIST Compatibility View (Deprecated) [page 2009]
The SYSUSERAUTH view is provided for compatibility with older versions of the software.

SYSUSERPERM Compatibility View (Deprecated) [page 2009]
Each row of the SYSUSERPERM view describes one user ID.

SYSUSERPERMS Compatibility View (Deprecated) [page 2010]
Each row of the SYSUSERPERMS view describes one user ID. However, password information is not included. All users are allowed to read from this view.

1.7.3.1 SYSCOLLATION Compatibility View (Deprecated)

The SYSCOLLATION compatibility view contains the collation sequence information for the database. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

```
ALTER VIEW "SYS"."SYSCOLLATION"
  as select 1 as collation_id,
    DB_PROPERTY('Collation') as collation_label,
    DB_EXTENDED_PROPERTY('Collation','Description') as collation_name,
    cast(DB_EXTENDED_PROPERTY('Collation','LegacyData') as binary(1280)) as
collation_order
```

Related Information

List of Database Server Properties
DB_PROPERTY Function [System] [page 337]
DB_EXTENDED_PROPERTY Function [System] [page 329]

1.7.3.2 SYSCOLLATIONMAPPINGS Compatibility View (Deprecated)

The SYSCOLLATIONMAPPINGS compatibility view contains only one row with the database collation mapping. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

```
ALTER VIEW "SYS"."SYSCOLLATIONMAPPINGS"
  as select DB_PROPERTY('Collation') as collation_label,
    DB_EXTENDED_PROPERTY('Collation','Description') as collation_name,
    DB_PROPERTY('Charset') as cs_label,
    DB_EXTENDED_PROPERTY('Collation','ASESensitiveSortOrder') as so_case_label,
    DB_EXTENDED_PROPERTY('Collation','ASEInsensitiveSortOrder') as
so_caseless_label,
    DB_EXTENDED_PROPERTY('Charset','java') as jdk_label
```

Related Information

List of Database Server Properties
DB_PROPERTY Function [System] [page 337]
DB_EXTENDED_PROPERTY Function [System] [page 329]

1.7.3.3 SYSCOLUMN Compatibility View (Deprecated)

The SYSCOLUMN view is provided for compatibility with older versions of the software that offered a SYSCOLUMN system table.

However, the previous SYSCOLUMN table has been replaced by the ISYSTABCOL system table, and its corresponding SYSTABCOL system view. Use the SYSTABCOL system view instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLUMN"
  as select b.table_id,
    b.column_id,
    if c.sequence is null then 'N' else 'Y' endif as pkey,
    b.domain_id,
    b.nulls,
    b.width,
    b.scale,
    b.object_id,
    b.max_identity,
    b.column_name,
    r.remarks,
    b."default",
    b.user_type,
    b.column_type
    from SYS.SYSTABCOL as b
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
      left outer join SYS.ISYSIDXCOL as c on(b.table_id = c.table_id and
b.column_id = c.column_id and c.index_id = 0)
```

Related Information

SYSTABCOL System View [page 1953]
SYSREMARK System View [page 1936]
SYSIDXCOL System View [page 1918]

1.7.3.4 SYSFILE Compatibility View (Deprecated)

Each row in the SYSFILE system view describes a dbspace for a database. Every database consists of one or more dbspaces; each dbspace corresponds to an operating system file.

dbspaces are automatically created for the main database file, temporary file, transaction log file, and transaction log mirror file. Information about the transaction log, and transaction log mirror dbspaces does not appear in the SYSFILE system view.

```
ALTER VIEW "SYS"."SYSFILE"
  as select b.dbfile_id as file_id,
    if b.dbspace_id = 0 and b.dbfile_id = 0 then
      db_property('File')
    else
      if b.dbspace_id = 15 and b.dbfile_id = 15 then
```

```
        db_property('TempFileName')
      else
        b.file_name
      endif
    endif as file_name,
    a.dbspace_name,
    a.store_type,
    b.lob_map,
    b.dbspace_id
    from SYS.ISYSDBSPACE as a
      join SYS.ISYSDBFILE as b on(a.dbspace_id = b.dbspace_id)
```

Related Information

Predefined Dbspaces

1.7.3.5 SYSFKCOL Compatibility View (Deprecated)

Each row of SYSFKCOL describes the association between a foreign column in the foreign table of a relationship and the primary column in the primary table. This view is deprecated; use the SYSIDX and SYSIDXCOL system views instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFKCOL"
  as select a.table_id as foreign_table_id,
    a.index_id as foreign_key_id,
    a.column_id as foreign_column_id,
    a.primary_column_id
    from SYS.ISYSIDXCOL as a
      ,SYS.ISYSIDX as b
    where a.table_id = b.table_id
    and a.index_id = b.index_id
    and b.index_category = 2
```

Related Information

SYSIDX System View [page 1916]
SYSIDXCOL System View [page 1918]

1.7.3.6 SYSFOREIGNKEY Compatibility View (Deprecated)

The SYSFOREIGNKEY view is provided for compatibility with older versions of the software that offered a SYSFOREIGNKEY system table. However, the previous SYSFOREIGNKEY system table has been replaced by the ISYSFKEY system table, and its corresponding SYSFKEY system view, which you should use instead.

A foreign key is a relationship between two tables: the foreign table and the primary table. Every foreign key is defined by one row in SYSFOREIGNKEY and one or more rows in SYSFKCOL. SYSFOREIGNKEY contains general information about the foreign key while SYSFKCOL identifies the columns in the foreign key and associates each column in the foreign key with a column in the primary key of the primary table.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFOREIGNKEY"
  as select b.foreign_table_id,
    b.foreign_index_id as foreign_key_id,
    a.object_id,
    b.primary_table_id,
    p.root,
    b.check_on_commit,
    b.nulls,
    a.index_name as role,
    r.remarks,
    b.primary_index_id,
    a.not_enforced as fk_not_enforced,
    10 as hash_limit
    from(SYS.ISYSIDX as a left outer join SYS.ISYSPHYSIDX as p on(a.table_id =
p.table_id and a.phys_index_id = p.phys_index_id))
      left outer join SYS.ISYSREMARK as r on(a.object_id = r.object_id)
      ,SYS.ISYSFKEY as b
    where a.table_id = b.foreign_table_id
    and a.index_id = b.foreign_index_id
```

Related Information

SYSIDX System View [page 1916]
SYSPHYSIDX System View [page 1930]
SYSREMARK System View [page 1936]
SYSFKEY System View [page 1912]

1.7.3.7 SYSGROUP Compatibility View

There is one row in the SYSGROUP system view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

Column name	Data type	Description
group_id	UNSIGNED INT	The user number of the group.

Column name	Data type	Description
group_member	UNSIGNED INT	The user number of a member.

1.7.3.8 SYSGROUPS Compatibility View

There is one row in the SYSGROUPS view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSGROUPS"( group_name,
  member_name )
  as select g.user_name,u.user_name
    from SYS.ISYSROLEGRANT,SYS.ISYSUSER as g,SYS.ISYSUSER as u
    where ISYSROLEGRANT.role_id = g.user_id and ISYSROLEGRANT.grantee =
u.user_id and(
    u.user_name in( 'SYS_SPATIAL_ADMIN_ROLE' )
    or u.user_id <= 2147483648) and(
    g.user_type = (0x02|0x04|0x08)
    or g.user_name in( 'SYS','PUBLIC','dbo','diagnostics',
    'rs_systabgroup','SA_DEBUG','SYS_SPATIAL_ADMIN_ROLE' ) )
```

Related Information

SYSUSER System View [page 1965]
SYSGROUP Compatibility View [page 2003]

1.7.3.9 SYSINDEX Compatibility View (Deprecated)

The SYSINDEX view is provided for compatibility with older versions of the software that offered a SYSINDEX system table. However, the SYSINDEX system table has been replaced by the ISYSIDX system table, and its corresponding SYSIDX system view, which you should use instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSINDEX"
  as select b.table_id,
    b.index_id,
    b.object_id,
    p.root,
    b.dbspace_id,
    case b."unique"
    when 1 then 'Y'
    when 2 then 'U'
    when 3 then 'M'
```

```
    when 4 then 'N'
    when 5 then 'Y'
    else 'I'
    end as "unique",
    t.creator,
    b.index_name,
    r.remarks,
    10 as hash_limit,
    b.dbspace_id as file_id
    from(SYS.ISYSIDX as b left outer join SYS.ISYSPHYSIDX as p on(b.table_id =
p.table_id and b.phys_index_id = p.phys_index_id))
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
      ,SYS.ISYSTAB as t
    where t.table_id = b.table_id
    and b.index_category = 3
```

Related Information

SYSIDX System View [page 1916]
SYSPHYSIDX System View [page 1930]
SYSTABLE Compatibility View (Deprecated) [page 2006]
SYSREMARK System View [page 1936]

1.7.3.10 SYSINFO Compatibility View (Deprecated)

The SYSINFO view indicates the database characteristics, as defined when the database was created. It always contains only one row. This view is obtainable via built-in functions and is not kept in the catalog. Following is the definition for the SYSINFO view:

```
ALTER VIEW "SYS"."SYSINFO"( page_size,
  encryption,
  blank_padding,
  case_sensitivity,
  default_collation,
  database_version )
  as select db_property('PageSize'),
    if db_property('Encryption') <> 'None' then 'Y' else 'N' endif,
    if db_property('BlankPadding') = 'On' then 'Y' else 'N' endif,
    if db_property('CaseSensitive') = 'On' then 'Y' else 'N' endif,
    db_property('Collation'),
    NULL
```

Related Information

List of Database Server Properties
DB_PROPERTY Function [System] [page 337]
DB_EXTENDED_PROPERTY Function [System] [page 329]

1.7.3.11 SYSIXCOL Compatibility View (Deprecated)

Each row of the SYSIXCOL describes a column in an index, and is provided for compatibility with older versions of the software that offered a SYSIXCOL system table.

The SYSIXCOL system table has been replaced by the ISYSIDXCOL system table, and its corresponding SYSIDXCOL system view. You should switch to using the SYSIDXCOL system view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSIXCOL"
  as select a.table_id,
    a.index_id,
    a.sequence,
    a.column_id,
    a."order"
    from SYS.ISYSIDXCOL as a
      ,SYS.ISYSIDX as b
    where a.table_id = b.table_id
    and a.index_id = b.index_id
    and b.index_category = 3
```

Related Information

SYSIDX System View [page 1916]
SYSIDXCOL System View [page 1918]

1.7.3.12 SYSTABLE Compatibility View (Deprecated)

The SYSTABLE view is provided for compatibility with older versions of the software that offered a SYSTABLE system table. However, the SYSTABLE system table has been replaced by the ISYSTAB system table, and its corresponding SYSTAB system view, which you should use instead.

Each row of SYSTABLE view describes one table in the database.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSTABLE"
  as select b.table_id,
    b.file_id,
    b.count,
    0 as first_page,
    b.commit_action as last_page,
    COALESCE(ph.root,0) as primary_root,
    b.creator,
    0 as first_ext_page,
    0 as last_ext_page,
    b.table_page_count,
    b.ext_page_count,
    b.object_id,
    b.table_name,
```

```
    b.table_type_str as table_type,
    v.view_def,
    r.remarks,
    b.replicate,
    p.existing_obj,
    p.remote_location,
    'T' as remote_objtype,
    p.srvid,
    case b.server_type
    when 1 then 'SA'
    when 2 then 'IQ'
    when 3 then 'OMNI'
    else 'INVALID'
    end as server_type,
    10 as primary_hash_limit,
    0 as page_map_start,
    s.source,
    b."encrypted"
    from SYS.SYSTAB as b
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
      left outer join SYS.ISYSSOURCE as s on(b.object_id = s.object_id)
      left outer join SYS.ISYSVIEW as v on(b.object_id = v.view_object_id)
      left outer join SYS.ISYSPROXYTAB as p on(b.object_id = p.table_object_id)
      left outer join(SYS.ISYSIDX as i left outer join SYS.ISYSPHYSIDX as ph
on(i.table_id = ph.table_id
        and i.phys_index_id = ph.phys_index_id)) on(b.table_id = i.table_id and
i.index_category = 1
        and i.index_id = 0)
```

Related Information

SYSTAB System View [page 1950]
SYSREMARK System View [page 1936]
SYSSOURCE System View [page 1943]
SYSVIEW System View [page 1970]
SYSPROXYTAB System View [page 1934]
SYSIDX System View [page 1916]
SYSPHYSIDX System View [page 1930]

1.7.3.13 SYSUSERAUTH Compatibility View (Deprecated)

Each row of the SYSUSERAUTH view describes a user, without exposing their user ID and password hash. Instead, each user is identified by their user name.

You must have the SELECT ANY TABLE system privilege to access this view.

The SYSUSERAUTH view is provided for compatibility with older versions of the software. Use the SYSROLEGRANTS consolidated view instead.

The password column displays three asterisks (***) if a value is present in the database and NULL if no value is present.

Although the title of this view contains the word auth (for authorities), the security model is based on roles and privileges. The data in the view is therefore compiled using role information from the tables and views mentioned in the view definition.

```
ALTER VIEW "SYS"."SYSUSERAUTH"( "name",
  "password","resourceauth","dbaauth","scheduleauth","user_group" )
  as select
"SYSUSERPERM"."user_name","SYSUSERPERM"."password","SYSUSERPERM"."resourceauth","
SYSUSERPERM"."dbaauth","SYSUSERPERM"."scheduleauth","SYSUSERPERM"."user_group"
    from "SYS"."SYSUSERPERM"
```

Related Information

SYSROLEGRANTS Consolidated View [page 1990]
SYSROLEGRANT System View [page 1939]
SYSROLEGRANTEXT System View [page 1940]
SYSUSER System View [page 1965]

1.7.3.14 SYSUSERAUTHORITY Compatibility View (Deprecated)

The SYSUSERAUTHORITY view is provided for compatibility with older versions of the software. Use the SYSROLEGRANTS consolidated view instead.

Each row of SYSUSERAUTHORITY system view describes an authority granted to one user ID.

Although the title of this view contains the word authority, the security model is based on roles and privileges. The data in the view is therefore compiled using role information from the tables and views mentioned in the view definition.

```
ALTER VIEW "SYS"."SYSUSERAUTHORITY" as
  select ISYSROLEGRANT.grantee as user_id,
    sp_auth_sys_role_info.auth
    from SYS.ISYSROLEGRANT
      natural join dbo.sp_auth_sys_role_info()
    where ISYSROLEGRANT.grant_type <> (0x02|0x04) and
    not ISYSROLEGRANT.grantee = any(select sp_auth_sys_role_info.role_id from
dbo.sp_auth_sys_role_info()) union
  select ISYSUSER.user_id,
    cast('GROUP' as varchar(20)) as auth
    from SYS.ISYSUSER
    where ISYSUSER.user_name
in( 'SYS','PUBLIC','diagnostics','SYS_SPATIAL_ADMIN_ROLE','rs_systabgroup','SA_DE
BUG','dbo' ) union
  select ISYSUSER.user_id,
    cast('GROUP' as varchar(20)) as auth
    from SYS.ISYSUSER
    where ISYSUSER.user_type = (0x02|0x04|0x08) union
  select cast(opt.setting as unsigned integer) as user_id,
    cast('PUBLISH' as varchar(20)) as auth
    from SYS.ISYSOPTION as opt
    where opt."option" like '%db_publisher%' and opt.setting not like '%-1%'
```

Related Information

SYSROLEGRANTS Consolidated View [page 1990]
SYSROLEGRANT System View [page 1939]
SYSROLEGRANTEXT System View [page 1940]

1.7.3.15 SYSUSERLIST Compatibility View (Deprecated)

The SYSUSERAUTH view is provided for compatibility with older versions of the software.

Each row of the SYSUSERLIST view describes a user, without exposing their user_id and password. Each user is identified by their user name.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSERLIST"( name,
  resourceauth,dbaauth,scheduleauth,user_group )
  as select
SYSUSERPERM.user_name,SYSUSERPERM.resourceauth,SYSUSERPERM.dbaauth,SYSUSERPERM.sc
heduleauth,SYSUSERPERM.user_group
    from SYS.SYSUSERPERM
```

Related Information

SYSUSERPERM Compatibility View (Deprecated) [page 2009]

1.7.3.16 SYSUSERPERM Compatibility View (Deprecated)

Each row of the SYSUSERPERM view describes one user ID.

This view is deprecated because it only shows the authorities and permissions available in previous versions. Change your application to use the SYSROLEGRANTS consolidated view.

You must have the SELECT ANY TABLE system privilege to access this view.

The password column displays three asterisks (***) if a value is present in the database and NULL if no value is present. To see actual password information, see the SYSUSERPASSWORD system view.

The tables and columns that make up this view are provided in the SQL statement below.

```
ALTER VIEW "SYS"."SYSUSERPERM"
  as select "b"."user_id",
    "b"."object_id",
    "b"."user_name",
    if "b"."password" is null then null else '***' endif as "password",
    if "AA"."resourceauth" is not null and "AA"."resourceauth" > 0 then
      'Y' else 'N' endif as "resourceauth",
```

```
    if "AA"."dbaauth" is not null and "AA"."dbaauth" > 0 then
      'Y' else 'N' endif as "dbaauth",
    'N' as "scheduleauth",
    if exists(select * from "SYS"."ISYSOPTION" as "opt"
      where "opt"."option" like '%db_publisher%' and "opt"."setting" not like
'%-1'
      and "b"."user_id" = cast("opt"."setting" as integer)) then
      'Y' else 'N' endif as "publishauth",
    if "AA"."remotedbaauth" is not null and "AA"."remotedbaauth" > 0 then
      'Y' else 'N' endif as "remotedbaauth",
    if "b"."user_type" = (0x02|0x04|0x08) or "b"."user_name"
in( 'SYS','PUBLIC','diagnostics','SYS_SPATIAL_ADMIN_ROLE','rs_systabgroup','SA_DE
BUG','dbo' ) then
      'Y' else 'N' endif as "user_group",
    "r"."remarks"
    from "SYS"."ISYSUSER" as "b"
      left outer join "SYS"."ISYSREMARK" as "r" on("b"."object_id" =
"r"."object_id")
      left outer join(select "sum"(if "sp_auth_sys_role_info"."auth" =
'RESOURCE' then 1 else 0 endif) as "resourceauth",
        "sum"(if "sp_auth_sys_role_info"."auth" = 'DBA' then 1 else 0 endif) as
"dbaauth",
        "sum"(if "sp_auth_sys_role_info"."auth" = 'REMOTE DBA' then 1 else 0
endif) as "remotedbaauth",
        "ISYSROLEGRANT"."grantee"
        from "SYS"."ISYSROLEGRANT" natural join "dbo"."sp_auth_sys_role_info"()
        where "ISYSROLEGRANT"."grant_type" <> (0x02|0x04)
        and "sp_auth_sys_role_info"."auth" in( 'DBA','RESOURCE','REMOTE DBA' )
        group by "ISYSROLEGRANT"."grantee") as "AA"
      on("AA"."grantee" = "b"."user_id")
```

Related Information

SYSROLEGRANTS Consolidated View [page 1990]
SYSROLEGRANT System View [page 1939]
SYSROLEGRANTEXT System View [page 1940]
SYSUSER System View [page 1965]
SYSUSERPASSWORD System View [page 1967]

1.7.3.17 SYSUSERPERMS Compatibility View (Deprecated)

Each row of the SYSUSERPERMS view describes one user ID. However, password information is not included. All users are allowed to read from this view.

This view is deprecated because it only shows the authorities and permissions available in previous versions. Change your application to use the SYSROLEGRANTS consolidated view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSERPERMS"
  as select
SYSUSERPERM.user_id,SYSUSERPERM.user_name,SYSUSERPERM.resourceauth,SYSUSERPERM.db
aauth,
```

```
SYSUSERPERM.scheduleauth,SYSUSERPERM.user_group,SYSUSERPERM.publishauth,SYSUSERPE
RM.remotedbaauth,SYSUSERPERM.remarks
    from SYS.SYSUSERPERM
```

Related Information

SYSROLEGRANTS Consolidated View [page 1990]
SYSROLEGRANT System View [page 1939]
SYSROLEGRANTEXT System View [page 1940]
SYSUSER System View [page 1965]

1.7.4 Views for Transact-SQL Compatibility

The Adaptive Server Enterprise and SQL Anywhere system catalogs are different.

The Adaptive Server Enterprise system tables and views are owned by the user dbo, and exist partly in the master database, partly in the sybsecurity database, and partly in each individual database. The SQL Anywhere system tables and views are owned by the special user SYS and exist separately in each database.

To assist in preparing compatible applications, SQL Anywhere provides the following set of views owned by the special user dbo, which correspond to their Adaptive Server Enterprise counterparts. Where architectural differences make the contents of a particular Adaptive Server Enterprise table or view meaningless in a SQL Anywhere context, the view is empty, containing just the column names and data types.

View name	Description
syscolumns	One row for each column in a table or view, and for each parameter in a procedure.
syscomments	One or more rows for each view, rule, default, trigger, and procedure, giving the SQL definition statement.
sysindexes	One row for each clustered or nonclustered index, one row for each table with no indexes, and an additional row for each table containing text or image data.
sysobjects	One row for each table, view, procedure, rule, trigger default, log, or (in tempdb only) temporary object.
systypes	One row for each system-supplied or user-defined data type.
sysusers	One row for each user allowed in the database.
syslogins	One row for each valid user account.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP