SQL Anywhere® サーバー SQL リファレンス

バージョン 12.0.1

2012 年 1 月

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

バージョン 12.0.1
2012 年 1 月

Copyright © 2012 iAnywhere Solutions, Inc. Portions copyright © 2012 Sybase, Inc. All rights reserved.

iAnywhere との間に書面による合意がないかぎり、このマニュアルは現状のまま提供されるものであり、その使用または記載内容の誤りに対して一切の責任を負いません。

次の条件に従うかぎり、このマニュアルの一部または全体を使用、印刷、複製、配布することができます。1) マニュアルの一部または全体にかかわらず、ここに示したものとそれ以外のすべての版権と商標の表示をすべてのコピーに含めること。2) マニュアルに変更を加えないこと。3) iAnywhere 以外の人間がマニュアルの著者または情報源であるかのように示す一切の行為をしないこと。

iAnywhere®、Sybase®、http://www.sybase.com/detail?id=1011207 に示す商標は Sybase, Inc. またはその関連会社の商標です。® は米国での登録商標を示します。

このマニュアルに記載されているその他の会社名と製品名は各社の商標である場合があります。

目次

はじめに v

SQL 言語の要素 1

キーワード 1
識別子 4
文字列 5
定数 6
演算子 8
式 12
探索条件 34
特別値 61
変数 76
コメント 84

SQL データ型 85

文字データ型 85
数値データ型 93
通貨データ型 103
ビット配列データ型 104
日付と時刻データ型 105
バイナリデータ型 122
ドメイン 125
データ型変換 127
空間データ型 138

SQL 関数 139

関数のタイプ 139
関数 151

SQL 文 403

一般的な SQL 構文要素 403
SQL 構文の表記規則 404
文の適応性インジケーター 406
SQL 文 406

テーブル 993

システムテーブル 993
診断トレーシングテーブル 1008
その他のテーブル 1026

システムプロシージャー 1029

システムプロシージャーとファンクションの詳細の表示 1029
Web サービスシステムプロシージャー 1030
MAPI プロシージャーと SMTP プロシージャー 1030
Adaptive Server Enterprise のシステムプロシージャーとカタログプロシージャー 1033
システムプロシージャーのアルファベット順リスト 1035

ビュー 1253

システムビュー 1253
統合ビュー 1335
互換ビュー 1356
Transact-SQL 互換のビュー 1365

索引 1367

はじめに

このマニュアルでは、システムプロシージャーとカタログ (システムテーブルとビュー) に関する情報について説明します。また、SQL Anywhere での SQL 言語の実装 (探索条件、構文、データ型、関数) についても説明します。

SQL 言語の要素

キーワード

各 SQL 文には 1 つまたは複数のキーワードが含まれています。SQL 文のキーワードでは大文字と小文字を区別しませんが、このマニュアルではキーワードを大文字で表記します。

たとえば、次の文では SELECT と FROM がキーワードになります。

```
SELECT *
  FROM Employees;
```

次の文は、上の文と同等です。

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

キーワードの中には、二重引用符で囲まないと識別子として使用できないものがあります。このようなキーワードを、予約語と呼びます。二重引用符で囲む必要のないキーワード (DBA など) もあり、これらは予約語ではありません。

予約語

SQL キーワードには「予約語」として定義されているものがあります。SQL 文で予約語を識別子として使用するには、二重引用符、角カッコ、または逆引用符で囲みます。SQL 文で使用するキーワードは、すべてではありませんが多くが予約語になっています。たとえば、次の構文を使用して、SELECT テーブルの内容を取り出します。

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

SQL キーワードは大文字と小文字が区別されず、次の各語は大文字、小文字、またはその任意の組み合わせで使用されます。次の語のいずれかと、大文字／小文字の区別のみが違う文字列はすべて予約語となります。

また、non_keywords オプションを使用してキーワードをオフにすることもできます。

reserved_keywords オプションは、デフォルトで無効にされている個別のキーワードを有効にします。

Embedded SQL を使用している場合、データベースライブラリ関数 sql_needs_quotes を使用すると文字列に二重引用符が必要かどうかを判別できます。文字列が予約語であるか、または通常識別子に使用できない文字が含まれている場合は、文字列に二重引用符を付けます。

sa_reserved_words システムプロシージャーを使用すると、予約語のリストを取得できます。

SQL Anywhere で予約語になっている SQL キーワードは次のとおりです。

add	**all**	**alter**	**and**
any	**as**	**asc**	**attach**
backup	**begin**	**between**	**bigint**
binary	**bit**	**bottom**	**break**
by	**call**	**capability**	**cascade**
case	**cast**	**char**	**char_convert**
character	**check**	**checkpoint**	**close**
comment	**commit**	**compressed**	**conflict**
connect	**constraint**	**contains**	**continue**
convert	**create**	**cross**	**cube**
current	**current_timestamp**	**current_user**	**cursor**
date	**datetimeoffset**	**dbspace**	**deallocate**
dec	**decimal**	**declare**	**default**
delete	**deleting**	**desc**	**detach**
distinct	**do**	**double**	**drop**
dynamic	**else**	**elseif**	**encrypted**
end	**endif**	**escape**	**except**
exception	**exec**	**execute**	**existing**
exists	**externlogin**	**fetch**	**first**
float	**for**	**force**	**foreign**
forward	**from**	**full**	**goto**
grant	**group**	**having**	**holdlock**
identified	**if**	**in**	**index**
inner	**inout**	**insensitive**	**insert**

inserting	**install**	**instead**	**int**
integer	**integrated**	**intersect**	**into**
is	**isolation**	**join**	**kerberos**
key	**lateral**	**left**	**like**
limit	**lock**	**login**	**long**
match	**membership**	**merge**	**message**
mode	**modify**	**natural**	**nchar**
new	**no**	**noholdlock**	**not**
notify	**null**	**numeric**	**nvarchar**
of	**off**	**on**	**open**
openstring	**openxml**	**option**	**options**
or	**order**	**others**	**out**
outer	**over**	**passthrough**	**precision**
prepare	**primary**	**print**	**privileges**
proc	**procedure**	**publication**	**raiserror**
readtext	**real**	**reference**	**references**
refresh	**release**	**remote**	**remove**
rename	**reorganize**	**resource**	**restore**
restrict	**return**	**revoke**	**right**
rollback	**rollup**	**save**	**savepoint**
scroll	**select**	**sensitive**	**session**
set	**setuser**	**share**	**smallint**
some	**spatial**	**sqlcode**	**sqlstate**
start	**stop**	**subtrans**	**subtransaction**
synchronize	**table**	**temporary**	**then**

time	**timestamp**	**tinyint**	**to**
top	**tran**	**treat**	**trigger**
truncate	**tsequal**	**unbounded**	**union**
unique	**uniqueidentifier**	**unknown**	**unsigned**
update	**updating**	**user**	**using**
validate	**values**	**varbinary**	**varbit**
varchar	**variable**	**varying**	**view**
wait	**waitfor**	**when**	**where**
while	**window**	**with**	**within**
work	**writetext**	**xml**	

参照

- 「sql_needs_quotes 関数」『SQL Anywhere サーバー プログラミング』
- 「non_keywords オプション」『SQL Anywhere サーバー データベース管理』
- 「reserved_keywords オプション」『SQL Anywhere サーバー データベース管理』
- 「sa_reserved_words システムプロシージャー」1161 ページ

識別子

識別子は、ユーザー ID、テーブル、カラムなど、データベースのオブジェクト名を表します。

備考

識別子の最大長は 128 バイトです。次のいずれかの条件と一致している場合は、二重引用符、角カッコ、または逆引用符 (`...`) で識別子を囲みます。

●識別子にスペースが含まれる

●識別子の先頭文字がアルファベット文字ではない (以下を参照)

●識別子が予約語である

●識別子にアルファベット文字や数字以外の文字が含まれる

「アルファベット文字」には、アルファベット、アンダースコア文字 (_)、アットマーク (@)、シャープ記号 (#)、ドル記号 ($) が含まれます。データベースの照合順は、どの文字がアルファベットまたは数字として扱われるかを指定します。

次の文字は、識別子では使用できません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

●" (二重引用符)

●制御文字 (0x20 未満の文字)

●円記号

quoted_identifier データベースオプションが Off に設定されている場合には、SQL 文字列を二重引用符で区切る必要があります。この SQL 文字列は識別子としては使用できません。ただし角カッコまたは逆引用符は、quoted_identifier の設定にかかわらず、識別子の区切りとして使用できます。quoted_identifier オプションのデフォルトは、Open Client 接続と jConnect 接続の場合は Off、それ以外の場合は On に設定されています。

標準と互換性

- **SQL/2008**　128 文字までの識別子を作成する機能は、SQL/2008 標準のオプションの SQL 言語機能 F391 です。

参照

- 「予約語」1 ページ
- 「quoted_identifier オプション」『SQL Anywhere サーバー データベース管理』

例

次の識別子はすべて有効です。

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

文字列

文字列は、サイズが最高で 2 GB の文字シーケンスです。文字列は SQL で次の用途に使用されます。

●「文字列リテラル」。文字列リテラルとは、一重引用符 (') で囲まれ、シーケンスで並べられた文字のことです。文字列リテラルは特定の定数値を表し、文字で簡単に入力できない特殊文字のエスケープシーケンスを含めることができます。

●CHAR データ型または NCHAR データ型のカラム値または変数値。

●式の評価結果。

文字列の長さは 2 つの方法で測定できます。

- **バイト長**　バイト長は、文字列のバイト数です。
- **文字長**　文字長は、文字列の文字数です。また使用している文字セットによって変わります。

cp1252 などの SBCS の場合、バイト長と文字長は同じです。マルチバイト文字セットの場合、文字列のバイト長は文字長以上です。

参照

- 「文字列リテラル」7 ページ

定数

この項では、バイナリリテラルと文字列リテラルについて説明します。

バイナリリテラル

バイナリリテラルは、0 ～ 9 の数字、大文字と小文字の A ～ F で構成される 16 進文字です。バイナリデータをリテラルとして入力する場合は、データの前に 0x (1 つのゼロと 1 つの後続文字) を指定します。このプレフィクスの右側には、偶数の桁数を指定します。たとえば、39 の 16 進数は 0027 であるため、0x0027 のように表されます。

0x12345678 の形式の 16 進定数は、バイナリ文字列として処理されます。0x の後ろに桁数を無制限に追加できます。

バイナリリテラルは、バイナリ定数とも呼ばれます。SQL Anywhere では、バイナリリテラルをおすすめします。

16 進値との変換

CAST、CONVERT、HEXTOINT、INTTOHEX 関数を使用すると、バイナリ文字列を整数に変換できます。CAST 関数と CONVERT 関数は、16 進定数を TINYINT、符号付きと符号なしの 32 ビット整数、符号付きと符号なしの 64 ビット整数、NUMERIC などに変換します。HEXTOINT 関数は、16 進定数を符号付き 32 ビット整数にのみ変換します。

CAST 関数によって返される値が 8 桁を超えることはできません。8 桁を超える値では、エラーが返されます。8 桁未満の値の左側に 0 が追加されます。たとえば、次の引数は値 -2,147,483,647 を返します。

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

次の引数はエラーを返します。これは、符号付き 32 ビット整数として 10 桁の値を表現できないためです。

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

符号付き 32 ビット整数として値を表現できる場合、HEXTOINT 関数によって返される値は 8 桁を超えることができます。HEXTOINT 関数は、数値、大文字または小文字の A ～ F のみで構成される文字列リテラルまたは変数を受け入れます (0x プレフィクスが付いている場合、付いていない場合の両方)。16 進値の右から 8 桁目が数 8 ～ 9 か大文字または小文字の A ～ F であるか、その前の桁がすべて大文字または小文字の F の場合、その 16 進値は負の整数値になります。

次の引数は、値 -2,147,483,647 を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT HEXTOINT( '0xFF80000001' );
SELECT HEXTOINT( '0x80000001' );
SELECT HEXTOINT ( '0xFFFFFFFFFFFFFFFF80000001' );
```

次の引数はエラーを返します。これは、引数が符号付き 32 ビット整数として表現できない正の整数値を表しているためです。

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

参照

- 「CAST 関数 [データ型変換]」165 ページ
- 「CONVERT 関数 [データ型変換]」179 ページ
- 「HEXTOINT 関数 [データ型変換]」241 ページ
- 「INTTOHEX 関数 [データ型変換]」257 ページ

文字列リテラル

文字列リテラルとは、一重引用符 (') で囲まれ、シーケンスで並べられた文字のことです。たとえば、'Hello world' は CHAR 型の文字列リテラルです。バイト長は 11 であり、文字長も 11 です。

文字列リテラルは、文字列定数、リテラル文字列、または単に文字列とも呼ばれます。SQL Anywhere では、文字列リテラルをおすすめします。

引用符で囲んだ値の前に N を付けることによって、NCHAR 文字列リテラルを指定できます。たとえば、N'Hello world' は NCHAR 型の文字列リテラルです。バイト長は 11 であり、文字長も 11 です。NCHAR 文字列リテラル内のバイトは、データベースの CHAR 文字セットを使用して解釈され、その後 NCHAR に変換されます。構文 N'string' は、CAST('string' AS NCHAR) の短縮形式です。

エスケープシーケンス

場合によっては、通常の方法では入力できない文字を文字列リテラルに配置する必要があります。たとえば、制御文字 (改行文字など)、一重引用符 (それ以外の場合、文字列リテラルの末尾を示す)、16 進のバイト値などです。このような場合、エスケープシーケンスを使用します。

次に、文字列リテラルにエスケープシーケンスを使用する例を示します。

- 一重引用符は、文字列リテラルの開始と末尾を示す場合に使用されます。このため、文字列の一重引用符は、'John''s database' のように一重引用符を追加してエスケープする必要があります。

- 円記号 (¥) の後に n、x、X、¥ 以外の文字がある場合、それらは別々の文字と解釈されます。たとえば、¥q であれば、円記号と q が挿入されます。

  16 進のエスケープシーケンスは、文字かバイナリ値かに関係なく使用できます。16 進のエスケープシーケンスは、バックスラッシュ、x に続けて 2 桁の 16 進数を指定します。16 進数値は、CHAR と NCHAR 両方の文字列リテラルで CHAR 文字セットの文字として解釈されます。

単一のタブ文字として値を格納しない場合、値 ¥x09 は ¥¥x09 としてコード化してください。ただし、¥xyy は ¥xyy として格納されます。コードページ 1252 の例 '123¥x80' は、数字 1、2、3 に続けてユーロ通貨記号を表現しています。.

- 文字列内の円記号は、円記号を追加して次のようにエスケープしてください。'c:¥¥november'。また、パスの場合は、円記号の代わりにスラッシュ (/) を使用することもできます。'c:/november'。
- 文字列内に改行を表すには、円記号と n (¥n) を使用して 'First line:¥nSecond line:' と指定します。

NCHAR 文字列リテラルには、CHAR 文字列リテラルと同じ文字とエスケープシーケンスを使用できます。

文字列リテラルに直接入力できない Unicode 文字を使用する場合、UNISTR 関数を使用します。

参照

- 「UNISTR 関数 [文字列]」380 ページ

演算子

この項では、算術演算子、文字列演算子、ビット処理演算子について説明します。

一般的な演算子の優先度が適用されます。カッコ内の式が最初に評価され、続いて乗算と除算、最後に加算と減算が評価されます。その後、文字列の連結が行われます。

参照

- 「演算子の優先度」12 ページ
- 「探索条件」34 ページ

比較演算子

比較の構文は、次のとおりです。

expression comparison-operator expression

ここで、*comparison-operator* は、次のいずれかになります。

オペレーター	説明
=	等価
>	より大きい
<	より小さい

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オペレーター	説明
>=	大きいかまたは等価
<=	小さいかまたは等価
!=	等価ではない
<>	等価ではない
!>	以下
!<	以上

大文字と小文字の区別

デフォルトでは、SQL Anywhere データベースは大文字と小文字を区別しないで作成されます。比較処理は、該当のデータベースの文字の区別に合わせて実行されます。SQL Anywhere データベースでの大文字と小文字の区別は、データベースの作成時に -c オプションを使用して制御できます。

大文字と小文字の区別は、データベース作成時に設定されます。

> **注意**
> 文字列の比較では、大文字と小文字を区別するようにデータベースを作成しないかぎり、**大文字と小文字を区別しません**

後続ブランク

文字列を比較するときの SQL Anywhere の動作は、データベース作成時に設定されます。

参照

- 「大文字と小文字の区別」『SQL Anywhere サーバー SQL の使用法』
- 「比較の後続ブランクを無視」『SQL Anywhere サーバー SQL の使用法』

論理演算子

探索条件は、AND 演算子または OR 演算子で結合できます。また、NOT 演算子を使用して探索条件を否定したり、IS 演算子を使用して式が true、false、または unknown と評価されるかをテストしたりすることもできます。

- **AND 演算子**　AND 演算子は、次のように探索条件の間に配置されます。

  .. **WHERE** *condition1* **AND** *condition2*

  AND を使用すると、両方の条件が TRUE の場合、結合した条件は TRUE になります。条件のいずれかが FALSE の場合は FALSE、それ以外の場合は UNKNOWN になります。

- **OR 演算子**　OR 演算子は、次のように探索条件の間に配置されます。

  .. **WHERE** *condition1* **OR** *condition2*

  OR を使用すると、条件のいずれかが TRUE の場合、結合した条件は TRUE になります。両方の条件が FALSE の場合は FALSE、それ以外の場合は UNKNOWN になります。

- **NOT 演算子**　NOT 演算子は、次のように条件の前に配置され、条件を否定します。

  .. **WHERE NOT** *condition*

  condition が FALSE の場合、NOT 条件は TRUE です。*condition* が TRUE の場合は FALSE、*condition* が UNKNOWN の場合は UNKNOWN になります。

- **IS 演算子**　IS 演算子は、式とテストする真理値の間に配置されます。IS 演算子の構文は、次のとおりです。

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

  expression が指定の *truth-value* (TRUE、FALSE、UNKNOWN、NULL のいずれか) と評価されれば IS 条件は TRUE になります。それ以外の場合、値は FALSE です。

  たとえば、5*3=15 IS TRUE は、式 5*3=15 が TRUE と評価されるかどうかをテストします。

参照

- 「3 値的論理」59 ページ

算術演算子

expression + expression　加算。いずれかの式が NULL 値の場合、結果は NULL 値になります。

expression - expression　減算。いずれかの式が NULL 値の場合、結果は NULL 値になります。

-expression　反転。式が NULL 値の場合、結果は NULL 値になります。

expression * expression　乗算。いずれかの式が NULL 値の場合、結果は NULL 値になります。

expression / expression　除算。いずれかの式が NULL の場合、または 2 番目の式が 0 の場合、結果は NULL になります。

expression % expression　モジュロによる、2 つの整数での除算の余り (整数) の算出。たとえば、21 を 11 で割ると商は 1、余りは 10 なので、21 % 11 = 10 になります。

標準と互換性

- **SQL/2008**　モジュロ演算子としての % の使用は、ベンダー拡張です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

文字列演算子

expression || expression　文字列連結 (2 つのパイプ記号)。いずれかの文字列が NULL 値の場合、連結には空文字列として扱われます。

expression + expression　代替の文字列連結。+ 連結演算子を使用する場合は、暗黙的データ変換を行わないで、必ずオペランドを文字データ型に明示設定してください。

たとえば、次のクエリは整数値 579 を返します。

```
SELECT 123 + 456;
```

これに対し、次のクエリは文字列 123456 を返します。

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

CAST または CONVERT 関数を使用すると、データ型を明示的に変換できます。

標準と互換性

- **SQL/2008**　|| 演算子は、SQL/2008 の文字列連結演算子です。ただし、SQL 標準では、|| のいずれかのオペランドが NULL 値の場合、連結の結果も NULL となります。SQL Anywhere では、|| オペランドは NULL を空の文字列として扱います。

ビット処理演算子

以下の演算子は、SQL Anywhere の bit データ型、integer データ型 (bit、tinyint、SMALLINT などの変形をすべて含む)、バイナリ値、およびビット配列データ型で使用できます。

演算子	説明
&	ビット処理 AND
|	ビット処理 OR
^	ビット処理の排他 OR
~	ビット処理 NOT

ビット処理演算子 &、|、~ は、論理演算子 AND、OR、NOT で代用することはできません。

標準と互換性

- **SQL/2008**　ベンダー拡張。ビット処理演算子 (および BIT VARYING データ型と BIT データ型) は、SQL/1999 標準では SQL 言語機能 F511 としてサポートされていました。この機能は、SQL/2003 標準から完全に削除されました。

例

たとえば、次の文は適切なビットが設定されているローを選択します。

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

ジョイン演算子

注意
Transact-SQL 外部ジョイン演算子 *= と =* はサポートされなくなりました。Transact SQL の外部ジョインを使用するには、tsql_outer_joins データベースオプションを On に設定します。

SQL Anywhere では、Transact-SQL の外部ジョイン演算子である *= および =* の 2 つの比較演算子が追加でサポートされます。これらの演算子のいずれかが比較述部に使用されると、暗黙的な LEFT OUTER JOIN または RIGHT OUTER JOIN が指定されます。

参照

- 「tsql_outer_joins オプション」『SQL Anywhere サーバー データベース管理』
- 「Transact-SQL の外部ジョイン (*= or =*)」『SQL Anywhere サーバー SQL の使用法』

演算子の優先度

式における演算子の優先度は次のとおりです。リストの最上部にある演算子から順に評価されます。

1. 単項演算子 (1 つのオペランドを必要とする演算子)
2. &, |, ^, ~
3. *, /, %
4. +, -
5. ||
6. **not**
7. **and**
8. **or**

1 つの式に複数の演算子を使用する場合、かっこを使用して演算子の順序を明示的に指定することをおすすめします。

式

式は、評価して値を返すことのできる文です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

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

case-expression の代替形式 :
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**]
FROM *table-name*

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

備考

式は、さまざまな場所で使用されます。

式は、数種類の要素で構成されます。これらについては、関数や変数に関する項で説明します。

式を評価するには、データベースに接続する必要があります。

関連する動作

なし。

参照

- 「式内の定数」14 ページ
- 「特別値」61 ページ
- 「式内のカラム名」14 ページ
- 「 SQL 関数」139 ページ
- 「式内のサブクエリ」15 ページ
- 「探索条件」34 ページ
- 「SQL データ型」85 ページ
- 「変数」76 ページ
- 「CASE 式」16 ページ

標準と互換性

- 以降の項で説明する式の各クラスの説明を参照してください。

式内の定数

定数とは、数値または文字列リテラルです。文字列定数は、アポストロフィ ('一重引用符') で囲まれています。文字列内にアポストロフィを表すには、アポストロフィを 2 つ続けて使用します。

式内のカラム名

カラム名は、オプションの相関名の後に続く識別子です。相関名は、通常はテーブル名です。

カラム名に英字、数字、アンダースコア以外の文字が使用されている場合は、二重引用符 ("") で囲んでください。次の例は、有効なカラム名です。

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

参照

- 「識別子」4 ページ
- 「FROM 句」758 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

式内のサブクエリ

サブクエリとは、別の SELECT 文、INSERT 文、UPDATE 文、DELETE 文、または別のサブクエリにネストされた SELECT 文のことです。

サブクエリは、一致するローがない場合は NULL と評価されます。

SELECT 文は、カッコで囲み、1 つの SELECT リスト項目のみを含めます。式として使用すると、通常、サブクエリは 1 つの値だけを返します。

サブクエリは、カラム名を使用できれば、任意の位置で使用できます。たとえば、別の SELECT 文の SELECT リストでも、サブクエリを使用できます。

参照

- 「探索条件内のサブクエリ」36 ページ

IF 式

IF 式の構文は、次のとおりです。

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

この式は次のように返します。

- *condition* が TRUE の場合、IF 式は *expression1* を返します。
- *condition* が FALSE の場合、IF 式は *expression2* を返します。
- *condition* が FALSE で *expression2* がない場合、IF 式は NULL を返します。
- *condition* が UNKNOWN の場合、IF 式は NULL を返します。

expression1 は、*condition* が TRUE の場合にのみ評価されます。同様に、*expression2* は、*condition* が FALSE の場合にのみ評価されます。*expression1* と *expression2* は任意の式であり、*condition* は有効な検索条件です。

> **注意**
> IF 式は、IF 文と同じではありません。

標準と互換性

- **SQL/2008**　ベンダー拡張。SQL/2008 標準には、IF 式の代わりに使用できる NULLIF、COALESCE、および CASE の各式が定義されています。

参照

- 「IF 文」791 ページ
- 「探索条件」34 ページ
- 「NULL 特別値」67 ページ

CASE 式

CASE 式は条件付きの SQL 式を提供します。CASE 式は、式が使用できればどこでも使用できます。

CASE 式の構文は、次のとおりです。

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

CASE 句に続く式が WHEN 句に続く式と等しい場合、THEN 文の後の式に復帰します。それ以外の場合、ELSE 文があればそれに続く式に復帰します。

ELSE 句が存在しなく、*expression-1* が *expression-2...expression-n* のいずれの値とも一致しない場合、CASE 式は NULL を返します。

たとえば、次のコードでは CASE 式が SELECT 文の 2 番目の句として使用されています。

```
SELECT ID,
  ( CASE Name
    WHEN 'Tee Shirt' then 'Shirt'
    WHEN 'Sweatshirt' then 'Shirt'
    WHEN 'Baseball Cap' then 'Hat'
    ELSE 'Unknown'
  END ) as Type
FROM Products;
```

代替構文は、次のとおりです。

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

WHEN 句の後の探索条件が満たされた場合、THEN 文に続く式に復帰します。それ以外の場合、ELSE 文があればそれに続く式に復帰します。

たとえば、次の文では、CASE 式が SELECT 文の 3 番目の句として使用され、探索条件と文字列を関連付けています。

```
SELECT ID, Name,
  ( CASE
    WHEN Name='Tee Shirt' then 'Sale'
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
      WHEN Quantity >= 50  then 'Big Sale'
      ELSE 'Regular price'
    END ) as Type
  FROM Products;
```

省略形 CASE 式の NULLIF 関数

NULLIF 関数は、CASE 句を省略形で記述する方法の 1 つです。NULLIF の構文は、次のとおりです。

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

NULLIF は 2 つの式の値を比較します。1 番目の式と 2 番目の式が等しい場合、NULLIF は NULL を返します。1 番目の式と 2 番目の式が異なる場合、NULLIF は 1 番目の式を返します。

> **注意**
> CASE 句の構文と CASE 式の構文を混同しないでください。

標準と互換性

- **SQL/2008**　CASE 式は、SQL/2008 標準のコア機能です。この標準では、文によって参照される式が、実行中の任意の時点で評価されることを許可しています。SQL Anywhere では、式の評価は、コンパイル時に決定できる定数値を除き、各 WHEN 句が評価されるときに、構文の順序に従って実行されます。

  CASE 式で END 以外に END CASE を使用することは、ベンダー拡張によってサポートされます。SQL/2008 標準の定義では、END は CASE 式で使用し、END CASE は CASE 句で使用します。

参照

- 「CASE 文」505 ページ

正規表現の概要

「正規表現」とは、文字列内で検索するパターンを定義する一連の文字、ワイルドカード、または演算子のことです。SQL Anywhere では、SELECT 文の WHERE 句での REGEXP 探索条件、または SIMILAR TO 探索条件の一部として、および REGEXP_SUBSTR 関数の引数として、正規表現がサポートされています。LIKE 探索条件では正規表現がサポートされていませんが、LIKE で指定できる一部のワイルドカードと演算子は、正規表現のワイルドカードと演算子に似ています。

次の SELECT 文は、正規表現 ((K|C[^h])%) を使用して、Contacts テーブルを検索し、K または C で始まる (ただし Ch は除く) 姓を持つ連絡先を返します。

```
SELECT Surname, GivenName
  FROM Contacts
  WHERE Surname SIMILAR TO '(K|C[^h])%';
```

正規表現では、次に説明するように追加構文を組み込んで、グループ化、量指定、アサーション、代替を指定できます。

- **グループ化**　グループ化を使用すると、正規表現の一部をグループ化していくつかの追加一致基準に適用できます。たとえば、'(abc){2}' は abcabc に一致します。

  グループ化を使用すると、式の各部が評価される順序の制御もできます。たとえば、'ab(cdcd)' では最初に cdcd の出現を検索し、次に、cdcd のインスタンスの前に ab があるかどうかを評価します。

- **量指定**　量指定を使用すると、式の先行部分が出現する回数を制御できます。たとえば、疑問符 (?) は、その前の文字の 0 または 1 つのインスタンスに一致する量指定子です。このため、'honou?r' は honor と honour の両方に一致します。

- **アサーション**　通常、パターンの検索ではそのパターンが返されます。アサーションを使用すると、返される表現にパターンに含めることなく、パターンの存在をテストできます。たとえば、'SQL(?= Anywhere)' は、SQL の後ろに 1 つのスペースと Anywhere が続く SQL にのみ一致します。

- **代替**　代替を使用すると、前のパターンが検出されない場合に検索する代替パターンを指定できます。代替パターンは左から右の順に評価され、最初の一致で検索が停止します。たとえば、'col(o|ou)r' は、color のインスタンスを検索します。インスタンスが検出されない場合、代わりに colour が検索されます。

参照

- 「正規表現の構文」18 ページ
- 「探索条件」34 ページ
- 「LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件」40 ページ
- 「REGEXP 探索条件」47 ページ
- 「SIMILAR TO 探索条件」48 ページ
- 「REGEXP_SUBSTR 関数 [文字列]」314 ページ

正規表現の構文

正規表現は、SIMILAR TO 検索条件、REGEXP 検索条件、および REGEXP_SUBSTR 関数でサポートされています。SIMILAR TO での正規表現の構文は ANSI/ISO SQL 標準に準拠しています。REGEXP と REGEXP_SUBSTR での正規表現の構文とサポートは Perl 5 に準拠しています。

正規表現は、REGEXP と SIMILAR TO では**文字列**の一致に使用されますが、REGEXP_SUBSTR では**部分文字列**の一致に使用されます。REGEXP と SIMILAR TO で部分文字列の一致を実現するには、一致させようとしているパターンの前後にワイルドカードを指定する方法があります。たとえば、REGEXP '.*car.*' は car、carwash、vicar に一致します。また、REGEXP_SUBSTR 関数を使用するようにクエリを書き直すことも可能です。

SIMILAR TO での正規表現を使用した一致では、大文字と小文字、およびアクセント記号が区別されます。REGEXP と REGEXP_SUBSTR は、データベースの大文字と小文字、およびアクセント記号の影響を受けません。

正規表現：メタ文字

メタ文字とは、正規表現内で特別な意味を持つ記号または文字のことです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メタ文字の扱いは次の条件で決まります。

●正規表現が使用されているのが、SIMILAR TO 探索条件または REGEXP 探索条件であるか、REGEXP_SUBSTR 関数であるか

●メタ文字が使用されているのが、正規表現の含まれている文字クラス内部かどうか

この先を読み進む前に、「文字クラス」の定義を理解しておく必要があります。文字クラスは角カッコで囲まれた文字のセットで、文字列中の文字との一致対象になります。たとえば、構文 SIMILAR TO 'ab[1-9]' の [1-9] は文字クラスで、1 ～ 9 のいずれかの数字に一致します。正規表現でのメタ文字の扱いは、メタ文字が文字クラス内にあるかどうかで異なります。具体的には、文字クラス内に配置されたメタ文字は、そのほとんどがメタ文字としてではなく通常の文字として扱われます。

SIMILAR TO の場合 (のみ)、メタ文字 *、?、+、_、|、(、)、{ は、文字クラス内でエスケープする必要があります。

文字クラスにリテラルのマイナス記号 (-)、脱字記号 (^)、または右側の角カッコ (]) を含めるには、これらの文字をエスケープする必要があります。

サポートされている正規表現メタ文字を次に示します。ほとんどすべてのメタ文字は、SIMILAR TO、REGEXP、および REGEXP_SUBSTR で同じように扱われます。

文字	その他の情報
[および]	左および右角カッコは、「文字クラス」の指定に使用されます。文字クラスとは、一致させる文字のセットです。 ハイフン (-) と脱字記号 (^) を除き、文字クラス内に指定されたメタ文字 (* など) や量指定子 ({m} など) は特別な意味を持たず、実際の文字として評価されます。 SQL Anywhere では、POSIX 文字クラスなどの部分文字クラスもサポートしています。
*	アスタリスクは、1 文字との 0 回以上の一致に使用できます。たとえば、REGEXP '.*abc' は、任意の文字列で始まって abc で終わる文字列と一致します。したがって、aabc、xyzabc、abc に一致しますが、bc や abcc には一致しません。
?	疑問符は、1 文字との 0 回または 1 回の一致に使用できます。たとえば、'colou?r' は color と colour に一致します。

文字	その他の情報
+	プラス記号は、1 文字との 1 回以上の一致に使用できます。たとえば、'bre+' は bre と bree に一致しますが、br には一致しません。
-	文字クラス内でハイフンを使用すると、範囲を指定できます。たとえば、REGEXP '[a-e]' は a、b、c、d、e に一致します。
%	パーセント記号は、SIMILAR TO で、任意の長さの文字列との一致に使用できます。 パーセント記号は、REGEXP および REGEXP_SUBSTR ではメタ文字として認識されません。指定されると、パーセント記号 (%) に一致します。
_ (アンダースコア文字)	アンダースコア文字は、SIMILAR TO で、1 文字との一致に使用できます。 アンダースコアは、REGEXP および REGEXP_SUBSTR ではメタ文字として認識されません。指定されると、アンダースコア (_) に一致します。
|	パイプ記号は、文字列の一致における代替パターンの指定に使用できます。パイプ記号で区切られたパターンを持つ文字列において、パイプ記号は OR と解釈され、一致は一番左のパターンから始まり、最初の一致で停止します。したがって、パターンは重要な順にリストしてください。代替パターンはいくつでも指定できます。
(および **)**	左および右丸カッコは、正規表現の一部のグループ化に使用されるメタ文字です。たとえば、(ab)* は、0 回以上の ab の繰り返しに一致します。数値式と同様、グループ化を使用すると、正規表現の各部が評価される順序を制御できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

文字	その他の情報
{ および }	左および右中カッコは、「量指定子」の指定に使用されるメタ文字です。量指定子では、一致を構成するためにパターンが繰り返される回数を指定します。次に例を示します。 ● **{*m*}**　ある 1 つの文字に正確に *m* 回一致します。たとえば、'519-[0-9]{3}-[0-9]{4}' は、市外局番 519 の電話番号に一致します (データが構文に定義されるフォーマットの場合)。 ● **{*m*,}**　ある 1 つの文字に少なくとも *m* 回一致します。たとえば、'[0-9]{5,}' は、5 桁以上の数字の文字列に一致します。 ● **{*m*,*n*}**　ある 1 つの文字に少なくとも *m* 回、ただし *n* 回以下、一致します。たとえば、SIMILAR TO '_{5,10}' は、5 ～ 10 文字の文字列に一致します。
¥	円記号は、メタ文字のエスケープ文字として使用されます。また、メタ文字ではない文字のエスケープにも使用できます。

文字	その他の情報
^	REGEXP と REGEXP_SUBSTR では、文字クラス外にある脱字記号は文字列の先頭に一致します。たとえば、'^[hc]at' は、文字列の先頭にある hat と cat にのみ一致します。 文字クラス内で使用された場合は、次のような動作になります。 ● **REGEXP および REGEXP_SUBSTR**　脱字記号が文字クラスの先頭文字である場合は、その文字セットに含まれていない文字に一致します。たとえば、REGEXP '[^abc]' は、a、b、c を除く任意の 1 文字に一致します。 脱字記号が角カッコ内の先頭文字ではない場合は、脱字記号に一致します。たとえば、REGEXP_SUBSTR '[a-e^c]' は a、b、c、d、e、^ に一致します。 ● **SIMILAR TO**　SIMILAR TO では、脱字記号は減算演算子として扱われます。たとえば、SIMILAR TO '[a-e^c]' は a、b、d、e に一致します。
$	REGEXP と REGEXP_SUBSTR で使用されると、文字列の末尾に一致します。たとえば、REGEXP 'cat$' は cat に一致し、catfish には一致しません。
.	REGEXP と REGEXP_SUBSTR で使用されると、任意の 1 文字に一致します。たとえば、REGEXP 'a.cd' は、a で始まり cd で終わる 4 文字の文字列に一致します。 SIMILAR TO で使用されると、ピリオド (.) に一致します。
:	コロンは、サブ文字クラスを指定するために文字セット内で使用されます。たとえば、'[[:alnum:]]' のように指定します。

正規表現：特殊部分文字クラス

「部分文字クラス」は、より大きな文字クラス内に埋め込まれた特殊文字クラスです。一致する文字のセットを定義するカスタム文字クラスのほかに (たとえば、[abxq4] は一致文字のセットを a、b、x、q、4 に制限します)、SQL Anywhere では、大部分の POSIX 文字クラスなどの部分

文字クラスをサポートしています。たとえば、[[:alpha:]] はすべての大文字と小文字のセットを表します。

REGEXP 探索条件と REGEXP_SUBSTR 関数では、次の表に示す構文の表記規則がすべてサポートされていますが、SIMILAR TO 検索式ではサポートされていません。SIMILAR TO でサポートされている表記規則は、「SIMILAR TO」の欄に Y と示されています。

REGEXP 内と REGEXP_SUBSTR 関数を使用する場合は、脱字記号を使用して部分文字クラスを否定できます。たとえば、[[:^alpha:]] は、アルファベット文字を除くすべての文字のセットに一致します。

部分文字クラス	その他の情報	SIMILAR TO
[:alpha:]	現在の照合ではアルファベットの大文字と小文字に一致します。たとえば、'[0-9]{3}[[:alpha:]]{2}' は、3 つの数字とそれに続く 2 つの文字に一致します。	Y
[:alnum:]	現在の照合では数字およびアルファベットの大文字と小文字に一致します。たとえば、'[[:alnum:]]+' は、1 つ以上の英数字の文字列に一致します。	Y
[:digit:]	現在の照合では数字に一致します。たとえば、'[[:digit:]-]+' は、1 つ以上の数字またはダッシュの文字列に一致します。同様に、'[^[:digit:]-]+' は、数字またはダッシュ以外の 1 文字以上の文字列に一致します。	Y
[:lower:]	現在の照合ではアルファベットの小文字に一致します。たとえば、A は大文字なので、'[[:lower:]]' は A に一致しません。	Y

部分文字クラス	その他の情報	SIMILAR TO
[:space:]	1 つのブランク (' ') に一致します。たとえば、次の文は、Contacts.City から 2 語で構成される名前の任意の都市を検索します。 SELECT City FROM Contacts WHERE City REGEXP '.*[[:space:]].*';	Y
[:upper:]	現在の照合ではアルファベットの大文字に一致します。たとえば、'[[:upper:]ab]' は a または b の大文字のどれかに一致します。	Y
[:whitespace:]	スペース、タブ、フォームフィード、キャリッジリターンなどのホワイトスペース文字に一致します。	Y
[:ascii:]	7 ビット ASCII 文字 (0 ～ 127 の序数) に一致します。	
[:blank:]	ブランクスペースまたは水平タブに一致します。 [[:blank:]] は [¥t] と同義です。	
[:cntrl:]	32 未満の序数または文字値 127 の ASCII 文字 (制御文字) に一致します。制御文字には、改行文字、フォームフィード、バックスペースなどが含まれます。	
[:graph:]	出力された文字に一致します。 [[:graph:]] は [[:alnum:] [:punct:]] と同義です。	

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

部分文字クラス	その他の情報	SIMILAR TO
[:print:]	出力された文字とスペースに一致します。 [[:print:]] は [[:graph:] [:whitespace:]] と同義です。	
[:punct:]	!"#$%&'()*+,-./:;<=>?@[¥]^_`{|}~ のいずれかに一致します。 [:punct:] の部分文字列クラスには、現在の照合で使用できる ASCII 以外の句読表記文字が含まれない場合があります。	
[:word:]	現在の照合ではアルファベット文字、数字、またはアンダースコア文字に一致します。 [[:word:]] は [[:alnum:]_] と同義です。	
[:xdigit:]	文字クラス [0-9A-Fa-f] に含まれる文字に一致します。	

正規表現：サポートされているその他の構文表記規則

次の構文表記規則が REGEXP 探索条件と REGEXP_SUBSTR 関数でサポートされており、円記号がエスケープ文字と認識されます。**これらの表記規則は、SIMILAR TO 検索式ではサポートされません。**

正規表現の構文	名前と意味
¥0 *xxx*	値が ¥0*xxx* の文字に一致します。ここで、*xxx* は任意の 8 進数の並びであり、0 はゼロです。たとえば、¥0134 は円記号に一致します。
¥a	ベル文字に一致します。
¥A	文字セットの外側で使用し、文字列の先頭に一致します。 文字セットの外側で使用する ^ と同義です。
¥b	バックスペース文字に一致します。

正規表現の構文	名前と意味
¥B	円記号 (¥) に一致します。
¥c *X*	指定された制御文字に一致します。たとえば、ctrl-Z. の場合は ¥cZ です。
¥d	現在の照合では数字に一致します。たとえば、次の文は、Contacts.Phone から 00 で終わるすべての電話番号を検索します。 SELECT Surname, Surname, City, Phone FROM Contacts WHERE Phone REGEXP '¥¥d{8}00'; ¥d は、文字クラスの内側と外側の両方で使用可能で、[[:digit:]] と同義です。
¥D	数字以外のものに一致します。これは、¥d の反対です。 ¥D は、文字クラスの内側と外側の両方で使用可能で、[^[:digit:]] と同義です。 否定の省略形を角カッコの内側で使用する場合は、注意が必要です。[¥D¥S] と [^¥d¥s] は同じではありません。後者は、数字またはホワイトスペース以外のすべての文字に一致します。このため、後者は x に一致しますが 8 には一致しません。一方、前者は、数字以外の文字またはホワイトスペース以外の文字のいずれかに一致します。数字はホワイトスペースではなく、ホワイトスペースは数字ではないため、[¥D¥S] は、任意の文字、数字、ホワイトスペース、その他に一致します。
¥e	エスケープ文字に一致します。
¥E	¥Q によって開始されたメタ文字を非メタ文字として扱う処理を終了します。
¥f	フォームフィードに一致します。
¥n	改行に一致します。
¥Q	¥E の箇所までは、すべてのメタ文字を非メタ文字として扱います。たとえば、¥Q[$¥E は ¥[¥$ と同義です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

正規表現の構文	名前と意味
¥r	キャリッジリターンに一致します。
¥s	スペースまたはホワイトスペースとして扱われる文字に一致します。たとえば、次の文は、Products.ProductName から名前に 1 つ以上のスペースが含まれるすべての製品名を返します。 SELECT Name FROM Products WHERE Name REGEXP '.*¥¥s.*' ¥s は、文字クラスの内側と外側の両方で使用可能で、[[:whitespace:]] と同義です。
¥S	ホワイトスペース以外の文字に一致します。これは ¥s の反対であり、[^[:whitespace:]] と同義です。 ¥S は、文字クラスの内側と外側の両方で使用できます。 否定の省略形を角カッコの内側で使用する場合は、注意が必要です。[¥D¥S] と [^¥d¥s] は同じではありません。後者は、数字またはホワイトスペース以外のすべての文字に一致します。このため、後者は x に一致しますが 8 には一致しません。一方、前者は、数字以外の文字またはホワイトスペース以外の文字のいずれかに一致します。数字はホワイトスペースではなく、ホワイトスペースは数字ではないため、[¥D¥S] は、任意の文字、数字、ホワイトスペース、その他に一致します。
¥t	水平タブに一致します。
¥v	垂直タブに一致します。

正規表現の構文	名前と意味
¥w	現在の照合ではアルファベット文字、数字、またはアンダースコアに一致します。たとえば、次の文は、Contacts.Surname から 7 文字長の英数字で構成されるすべての姓を返します。 SELECT Surname FROM Contacts WHERE Surname REGEXP '¥¥w{7}'; ¥w は、文字クラスの内側と外側の両方で使用できます。 [[:alnum:]_]. と同義です。
¥W	現在の照合ではアルファベット文字、数字、またはアンダースコア以外のものに一致します。これは ¥w の反対であり、[^[:alnum:]_] と同義です。 この正規表現は、文字クラスの内側と外側の両方で使用できます。
¥x *hh*	値が 0x*hh* の文字に一致します。ここで、*hh* は 2 桁以下の 16 進数です。たとえば、¥x2D はハイフンと同義です。 ¥x{*hh*} と同義です。
¥x{ *hhh* **}**	値が 0x*hhh* の文字に一致します。ここで、*hhh* は 8 桁以下の 16 進数です。
¥z および **¥Z**	文字列の末尾の位置 (文字ではない) に一致します。 $ と同義です。

正規表現：アサーション

アサーションは、条件が true かどうかをテストし、文字列内の一致が開始される位置に影響します。アサーションは文字を返しません。最終的な一致にはアサーションのパターンは含まれません。これらのアサーションは、REGEXP 探索条件と REGEXP_SUBSTR 関数でサポートされています。これらの表記規則は、SIMILAR TO 検索式ではサポートされません。

REGEXP_SUBSTR で文字列を分割する場合は、先読みアサーションと後読みアサーションが便利です。たとえば、次の文を実行すると、Customers テーブルの Address カラムに含まれる街路名のリストを (街路番号を付けずに) 返すことができます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

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

次の例は、正規表現を使用してパスワードが特定の規則を満たしていることを確認する場合を示します。次の文に類似したゼロ幅アサーションを使用できます。

```
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
```

次の条件を満たす場合、パスワードは有効です。

- *password* に 1 つ以上の数字が含まれている ([[:digit:]] を使用したゼロ幅の正のアサーション)
- *password* に 2 つ以上のアルファベット文字が含まれている ([[:alpha:]].*[[:alpha:]] を使用したゼロ幅の正のアサーション)
- *password* に英数字またはアンダースコア文字のみが含まれている ([[:word:]])
- *password* が 4 文字以上で 12 文字以下である ({4,12})

次の表は、SQL Anywhere でサポートされているアサーションを示します。

構文	意味
(?= *pattern* **)**	**正の先読みゼロ幅アサーション**　一致文字列に *pattern* を含めないで、文字列内の現在位置のすぐ後ろに *pattern* があるかどうかを調べます。'A(?=B)' は、後ろに B が続く A に一致します。B は一致に含まれません。 たとえば、SELECT REGEXP_SUBSTR('in new york city', 'new(?=¥¥syork)'); は、部分文字列 new の後ろに ' york' (スペースの後ろに york があります) が続く場合に、new を返します。
(?! *pattern* **)**	**負の先読みゼロ幅アサーション**　文字列内の現在位置のすぐ後ろに *pattern* が**ない**かどうかを調べます。一致文字列に *pattern* は含まれません。したがって、'A(?!B)' は、後ろに B が続かない A に一致します。 たとえば、SELECT REGEXP_SUBSTR('new jersey', 'new(?!¥¥syork)'); は、部分文字列 new を返します。

構文	意味
(?<= *pattern* **)**	**正の後読みゼロ幅アサーション**　一致文字列に *pattern* を含めないで、文字列内の現在位置のすぐ前に *pattern* があるかどうかを調べます。したがって、'(?<=A)B' は、すぐ前に A がある B に一致します。A は一致に含まれません。 たとえば、SELECT REGEXP_SUBSTR('new york', '(?<=new¥¥s)york'); は、部分文字列 york を返します。
(?<! *pattern* **)**	**負の後読みゼロ幅アサーション**　文字列内の現在位置のすぐ前に *pattern* が**ない**かどうかを調べます。一致文字列に *pattern* は含まれません。 たとえば、SELECT REGEXP_SUBSTR('about york', '(?<!new¥¥s)york'); は、部分文字列 york を返します。
(?> *pattern* **)**	**所有ローカル部分式**　*pattern* に一致する残りの文字列の最大プレフィクスにのみ一致します。 たとえば、'aa' REGEXP '(?>a*)a' では、(?>a*) は先行の a にのみ一致するのではなく、aa に一致 (つまり、条件の評価対象全体に一致) します。このため、'aa' REGEXP '(?>a*)a' は、false と評価されます。
(?: *pattern* **)**	**非取得ブロック**　これは、機能的には単なる *pattern* と同じであり、互換性を保つために提供されています。 たとえば、'bb' REGEXP '(?:b*)b' では、(?:b*) は bb に一致 (つまり、条件の評価対象全体に一致) します。ただし、所有ローカル部分式とは異なり、bb の最後の b は解放され、全体の一致が続行されます (つまり、非取得ブロックの外側に指定されている b との一致が可能)。 同様に、'a(?:bc|b)c' は abcc と abc に一致します。abc との一致の場合は、bc の最後の c でバックトラックが実行され、グループの外側にある c が使用されて一致が成功します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文	意味
(?# *text* **)**	コメントに使用します。*text* の内容は無視されます。

参照

- 「正規表現の例」31 ページ
- 「REGEXP_SUBSTR 関数 [文字列]」314 ページ
- 「LIKE、REGEXP、SIMILAR TO：文字の比較における相違点」40 ページ
- 「正規表現：メタ文字」18 ページ
- 「正規表現：特殊部分文字クラス」22 ページ

正規表現の例

次の表は、正規表現の使用例を示します。これらの例はすべて REGEXP で機能します。一部は SIMILAR TO でも機能し、その場合は「例」の欄にその旨が示されています。結果は、使用する探索条件によって異なります。SIMILAR TO で機能する正規表現の場合は、大文字と小文字やアクセント記号を区別するかどうかによって、結果はさらに異なります。

これらの例をリテラル文字列 ('.+@.+¥¥..+' など) で使用する場合は、円記号を 2 つ重ねる必要があるので、注意してください。

例	一致のサンプル
「クレジットカード番号 (REGEXP のみ)：」 ビザ： 4[0-9]{3}¥s[0-9]{4}¥s[0-9]{4}¥s[0-9]{4} マスターカード： 5[0-9]{3}¥s[0-9]{4}¥s[0-9]{4}¥s[0-9]{4} アメリカンエキスプレス： 37[0-9]{2}¥s[0-9]{4}¥s[0-9]{4}¥s[0-9]{4} ディスカバー： 6011¥s[0-9]{4}¥s[0-9]{4}¥s[0-9]{4}	一致するもの (ビザ)： 4123 6453 2222 1746 一致しないもの (ビザ)： 3124 5675 4400 4567, 4123-6453-2222-1746 同様に、マスターカードでは、5 から始まり、4 つの数字のサブセット間にスペースがある 16 個の数字のセットに一致します。アメリカンエキスプレスとディスカバーは同じですが、それぞれ 37 と 6011 から始まります。
「日付 (REGEXP および SIMILAR TO)：」 ([0-2][0-9]|30|31)/(0[1-9]|1[0-2])/[0-9]{4}	一致するもの：31/04/1999、15/12/4567 31/04/1999, 15/12/4567 一致しないもの：31/4/1999、31/4/99、1999/04/19、42/67/25456 31/4/1999, 31/4/99, 1999/04/19, 42/67/25456

例	一致のサンプル
「Windows の絶対パス (REGEXP のみ)：」 ([A-Za-z]:|¥¥)¥¥[[:alnum:][:whitespace:]!"#$%&'()+,-.¥¥;=@¥[¥]^_`{}~.]*	一致するもの：¥¥server¥share¥file 一致しないもの：¥directory¥directory2、/directory2
「電子メールアドレス (REGEXP のみ)：」 [[:word:]¥-.]+@[[:word:]¥-.]+¥.[[:alpha:]]{2,3}	一致するもの：abc.123@def456.com、_123@abc.ca 一致しないもの：abc@dummy、ab*cd@efg.hijkl
「電子メールアドレス (REGEXP のみ)：」 .+@.+¥..+	一致するもの：*@qrstuv@wxyz.12345.com、__1234^%@@abc.def.ghijkl 一致しないもの：abc.123.*&ca、^%abcdefg123
「HTML の 16 進カラーコード (REGEXP および SIMILAR TO)：」 [A-F0-9]{6}	一致するもの：AB1234、CCCCCC、12AF3B 一致しないもの：123G45、12-44-CC
「HTML の 16 進カラーコード (REGEXP のみ)：」 [A-F0-9]{2}¥s[A-F0-9]{2}¥s[A-F0-9]{2}	一致するもの：AB 11 00、CC 12 D3 一致しないもの：SS AB CD、AA BB CC DD、1223AB
「IP アドレス (REGEXP のみ)：」 ((2(5[0-5]|[0-4][0-9])|1([0-9][0-9])|([1-9][0-9])|[0-9])¥.){3}(2(5[0-5]|[0-4][0-9])|1([0-9][0-9])|([1-9][0-9])|[0-9])	一致するもの：10.25.101.216 10.25.101.216 一致しないもの：0.0.0、256.89.457.02 0.0.0, 256.89.457.02
「Java コメント (REGEXP のみ)：」 /¥*.*¥*/|//[^¥n]*	一致するもの：/* と */ の間にある Java コメント、または先頭に // が付く 1 行のコメント 一致しないもの：a=1
「通貨 (REGEXP のみ)：」 (¥+|-)?¥$[0-9]*¥.[0-9]{2}	一致するもの：$1.00、-$97.65 $1.00, -$97.65 一致しないもの：$1、1.00$、$-75.17 $1, 1.00$, $-75.17
「正の値、負の値、小数値 (REGEXP のみ)：」 (¥+|-)?[0-9]+(¥.[0-9]+)?	一致するもの：+41、-412、2、7968412、41、+41.1、-3.141592653 +41, -412, 2, 7968412, 41, +41.1, -3.141592653 一致しないもの：++41、41.1.19、-+97.14 ++41, 41.1.19, -+97.14

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例	一致のサンプル
「パスワード (REGEXP および SIMILAR TO)：」 [[:alnum:]]{4,10}	一致するもの：abcd、1234、A1b2C3d4、1a2B3 一致しないもの：abc、*ab12、abcdefghijkl
「パスワード (REGEXP のみ)：」 [a-zA-Z]¥w{3,7}	一致するもの：AB_cd、A1_b2c3、a123_ 一致しないもの：*&^g、abc、1bcd
「電話番号 (REGEXP および 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})	一致するもの：519-883-6898、519 888 6898 519-883-6898, 519 888 6898 一致しないもの：888 6898、5198886898、519 883-6898 888 6898, 5198886898, 519 883-6898
「文章 (REGEXP のみ)：」 [A-Z0-9].*(¥.|¥?|!)	一致するもの：Hello, how are you? 一致しないもの：i am fine
「文章 (REGEXP のみ)：」 [[:upper:]0-9].*[.?!]	一致するもの：Hello, how are you? 一致しないもの：i am fine
「社会保障番号 (REGEXP および SIMILAR TO)：」 [0-9]{3}-[0-9]{2}-[0-9]{4}	一致するもの：123-45-6789 一致しないもの：123 45 6789、123456789、1234-56-7891
「URL (REGEXP のみ)：」 (http://)?www¥.[a-zA-Z0-9]+¥.[a-zA-Z]{2,3}	一致するもの：http://www.sample.com、www.sample.com 一致しないもの：http://sample.com、http://www.sample.comm

参照

- 「正規表現の構文」18 ページ
- 「LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件」40 ページ

式の互換性

SQL Anywhere では、アポストロフィで囲まれた文字列を定数の式、二重引用符で囲まれた文字列を区切り識別子 (データベースオブジェクト用の名前) とする SQL/2008 の規則を採用しています。

quoted_identifier オプション

SQL Anywhere には、区切り文字列の解釈方法を変更できる quoted_identifier オプションがあります。SQL Anywhere のデフォルトでは、quoted_identifier オプションは On に設定されています。

quoted_identifier オプションが Off の場合、SQL の予約語は識別子として使用できません。

オプションの設定

次の文では、quoted_identifier オプション設定を On に変更します。

```
SET quoted_identifier On;
```

次の文では、quoted_identifier オプション設定を Off に変更します。

```
SET quoted_identifier Off;
```

区切り文字列の互換性のある解釈

SQL Anywhere の各 DBMS で quoted_identifier オプションが同じ値に設定されていれば、SQL/2008 の規則またはデフォルトの Transact-SQL の規則のどちらでも使用できます。

例

quoted_identifier オプションを On (デフォルト設定) で動作するように選択した場合、SQL キーワード **user** を指定した次の文はどちらの DBMS でも有効です。

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

quoted_identifier オプションを off に設定する場合、次の文はどちらの DBMS でも有効です。次の例では、Chin は文字列であり、識別子ではありません。

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

参照

- 「quoted_identifier オプション」『SQL Anywhere サーバー データベース管理』
- 「予約語」1 ページ

探索条件

探索条件は、WHERE 句、HAVING 句、CHECK 句、ジョインの ON フレーズ、または IF 式に指定された基準です。探索条件は、述部とも言います。

構文

search-condition :
expression comparison-operator expression

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| *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 (** [**ONLY**] *type-name* ,... **)**
| *trigger-operation*

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

trigger-operation :
INSERTING
| **DELETING**
| **UPDATING** [**(** *column-name-string* **)**]
| **UPDATE(** *column-name* **)**

パラメーター

●ALL 探索条件

●ANY 探索条件と SOME 探索条件

●IS [NOT] DISTINCT FROM 探索条件

●BETWEEN 探索条件

●CONTAINS 探索条件

●EXISTS 探索条件

●LIKE 探索条件

●SIMILAR TO 探索条件

●REGEXP 探索条件

- IS OF *type-expression* と IS NOT OF *type-expression*
  このタイプ述部は、空間ジオメトリのサポート用に追加されましたが、既存のデータ型にも使用できます。

備考

探索条件は、テーブル内からローのサブセットを選択するか、または IF 文などの制御文でフローの制御を決めるために使用します。

SQL では、すべての条件が TRUE、FALSE、または UNKNOWN のいずれかに評価されます。これは、3 値的論理といいます。比較される値のいずれかが NULL の場合、比較結果は UNKNOWN になります。

比較結果が TRUE のみの場合、ローは探索条件を満たします。条件が UNKNOWN または FALSE のローは、探索条件を満たしません。

サブクエリは、多数の探索条件で使用される式の重要なクラスを構成します。

LIKE 探索条件、SIMILAR TO 探索条件、REGEXP 探索条件はよく似ています。

パーミッション

データベースに接続されている必要があります。

関連する動作

なし。

参照

- 「式」12 ページ
- 「ALL 探索条件」37 ページ
- 「ANY 探索条件と SOME 探索条件」38 ページ
- 「BETWEEN 探索条件」39 ページ
- 「CONTAINS 探索条件」51 ページ
- 「EXISTS 探索条件」57 ページ
- 「LIKE 探索条件」42 ページ
- 「SIMILAR TO 探索条件」48 ページ
- 「REGEXP 探索条件」47 ページ
- 「空間データ型構文の理解」『SQL Anywhere サーバー 空間データサポート』
- 「3 値的論理」59 ページ
- 「NULL 特別値」67 ページ
- 「探索条件内のサブクエリ」36 ページ
- 「LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件」40 ページ

探索条件内のサブクエリ

1 つのカラムと、0 もしくは 1 つのローを確実に返すサブクエリは、式の途中などカラム名を使用できる位置であれば SQL 文内のどこにでも使用できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

たとえば、サブクエリがローを 1 つだけ返すかぎりは、式を比較条件内のサブクエリと比較できます。サブクエリ (カラムは 1 つだけ) がローを 1 つ返す場合、そのローの値は式と比較されます。サブクエリがローを返さない場合、サブクエリの値は NULL です。

1 つのカラムと任意の数のローを返すサブクエリは、IN、ANY、ALL、SOME の各探索条件で使用できます。任意の数のカラムとローを返すサブクエリは、EXISTS 探索条件で使用できます。以降の項で、各探索条件について説明します。

標準と互換性

- **SQL/2008** スカラサブクエリの任意の式としての使用は、SQL/2008 標準のコア機能です。

参照

- 「比較演算子」8 ページ

ALL 探索条件

構文

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

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

備考

ALL 探索条件では、サブクエリの結果セットの値が空のセットだった場合、探索条件は TRUE と評価されます。それ以外の場合は、次に示すように、*expression* の値とサブクエリが返す結果セットに応じて、TRUE、FALSE、または UNKNOWN になります。

式の値	サブクエリが返す結果セットに 1 つ以上の NULL が含まれている場合	サブクエリが返す結果セットに NULL がない場合
NULL	UNKNOWN	UNKNOWN
NOT NULL	式の値と比較した結果が FALSE になる値がサブクエリの結果セットに 1 つでもあると、探索条件は FALSE と評価されます。それ以外の場合は、UNKNOWN になります。	式の値と比較した結果が FALSE になる値がサブクエリの結果セットに 1 つでもあると、探索条件は FALSE と評価されます。それ以外の場合は、TRUE になります。

標準と互換性

- **SQL/2008** コア機能。

ANY 探索条件と SOME 探索条件

構文

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

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

備考

キーワード ANY と SOME は同義です。

ANY 探索条件と SOME 探索条件では、サブクエリの結果セットが空のセットだった場合、探索条件は FALSE と評価されます。それ以外の場合は、次に示すように、*expression* の値とサブクエリが返す結果セットに応じて、TRUE、FALSE、または UNKNOWN になります。

式の値	サブクエリが返す結果セットに 1 つ以上の NULL が含まれている場合	サブクエリが返す結果セットに NULL がない場合
NULL	UNKNOWN	UNKNOWN
NOT NULL	式の値と比較した結果が TRUE になる値がサブクエリの結果セットに 1 つでもあると、探索条件は TRUE と評価されます。それ以外の場合は、UNKNOWN になります。	式の値と比較した結果が TRUE になる値がサブクエリの結果セットに 1 つでもあると、探索条件は TRUE と評価されます。それ以外の場合は、FALSE になります。

等号演算子のある ANY 探索条件または SOME 探索条件は、*expression* がサブクエリ結果のいずれかの値と等しい場合は TRUE、expression が NULL ではなく、サブクエリ結果のいずれの値とも等しくなく、結果セットに NULL が含まれていない場合は FALSE と評価されます。

注意
= **ANY** または = **SOME** を使用することは、IN キーワードを使用することと同等です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　コア機能。

IS DISTINCT FROM および IS NOT DISTINCT FROM 探索条件

構文

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

備考

IS DISTINCT FROM および IS NOT DISTINCT FROM 探索条件は、検索引数可能であり、TRUE または FALSE と評価されます。

IS NOT DISTINCT FROM 探索条件は、*expression1* が *expression2* と等しい場合、または両方の式が NULL の場合、TRUE と評価されます。これは、次のような 2 つの探索条件の組み合わせと同じです。

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

IS DISTINCT FROM 構文は、逆の意味を持ちます。つまり、IS DISTINCT FROM は、*expression1* が *expression2* と等しくなく、少なくとも一方の式が NULL ではない場合、TRUE と評価されます。これは、次の探索条件と同じです。

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

標準と互換性

- **SQL/2008**　IS [NOT] DISTINCT FROM 述部は、SQL/2008 標準に定義されています。IS DISTINCT FROM 述部は、SQL/2008 標準の機能 T151 の "DISTINCT predicate" です。IS NOT DISTINCT FROM 述部は、SQL/2008 標準の機能 T152 の "DISTINCT predicate with negation" です。

参照

- 「クエリの述部」『SQL Anywhere サーバー SQL の使用法』

BETWEEN 探索条件

構文

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

備考

BETWEEN 探索条件は、TRUE、FALSE、または UNKNOWN として評価できます。NOT キーワードがない場合、*expression* が *start-expression* と *end-expression* の間にあれば、探索条件は TRUE と評価されます。NOT キーワードを使用すると探索条件の意味が逆になりますが、UNKNOWN は変わりません。

BETWEEN 探索条件は、次の 2 つの不等式の組み合わせと等価です。

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

標準と互換性

- **SQL/2008**　コア機能。

LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件

REGEXP、LIKE、SIMILAR TO の各探索条件は、どれもパターンと文字列を一致させようとする点で似ています。また、これらの 3 つはすべて、文字列内の部分文字列ではなく、文字列全体に一致させようとします。

これらの 3 つの探索条件の基本的な構文は似ています。

expression search-condition pattern

LIKE、REGEXP、SIMILAR TO：*pattern* の定義における相違点

- REGEXP では、SIMILAR TO でサポートされる正規表現の構文のスーパーセットがサポートされています。また、他の製品と互換性を持たせるため、REGEXP 探索条件では構文拡張がいくつかサポートされています。また、REGEXP と SIMILAR TO は、デフォルトのエスケープ文字が異なり、アンダースコア (_)、パーセント (%)、および脱字記号 (^) の処理が異なります。REGEXP の動作は Perl 5 とほとんど一致します (ただし、Perl の構文と演算子がサポートされていない箇所を除く)。

- *pattern* の LIKE 構文は単純で、少数のワイルドカードがサポートされていますが、正規表現構文は完全にはサポートされていません。

- *pattern* の SIMILAR TO 構文では、ANSI/ISO SQL 標準に定義されている正規表現の構文を使用した強力なパターン一致を実行できます。

LIKE、REGEXP、SIMILAR TO：文字の比較における相違点

比較の実行において、REGEXP の動作は LIKE や SIMILAR TO とは異なります。REGEXP の比較では、データベースサーバーは「データベース文字セット」のコードポイント値を比較に使用します。この動作は、Perl などでの正規表現の実装と同じです。

LIKE と SIMILAR TO の場合、データベースサーバーは「データベース照合」の等価性とソート順を比較に使用します。この動作は、データベースが > や = などの比較演算子を評価する方法と同じです。

文字の比較方法が違うため、REGEXP と LIKE および SIMILAR TO とでは一致や範囲評価の結果が異なります。

- **一致における相違点**　REGEXP では、コードポイント値を使用するため、パターンに含まれるリテラルと一致するのは、厳密な意味で同じ文字の場合のみです。このため、REGEXP での一致は、データベース照合、大文字と小文字の区別、またはアクセント記号の区別といった要因には影響されません。たとえば、'a' との一致として 'A' が返されることはありません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

LIKE と SIMILAR TO ではデータベース照合が使用されるため、文字が等価かどうかを判断する際に、大文字と小文字やアクセント記号の区別に影響されます。たとえば、データベース照合で大文字と小文字やアクセント記号が区別されない場合、一致でも大文字と小文字やアクセント記号が区別されません。そのため、'a' との一致として 'A' が返されることがあり得ます。

● **範囲評価における相違点**　REGEXP では範囲評価にコードポイントが使用されるため、文字が範囲内と判断されるのは、その文字のコードポイント値が、範囲の最初と最後のコードポイント値と同じかその間にある場合になります。たとえば、比較 x REGEXP '[A-C]' は、単一文字 *x* について、CAST(x AS BINARY) >= CAST(A AS BINARY) AND CAST(x AS BINARY) <= CAST(C AS BINARY) と同義です。

LIKE と SIMILAR TO では、範囲評価に照合のソート順が使用されるため、文字が範囲内と判断されるのは、その文字の照合における位置が、範囲の最初と最後の文字の位置と同じかその間にある場合になります。たとえば、比較 x SIMILAR TO '[A-C]' (*x* は 1 文字) は x >= A AND x <= C と同義であり、比較演算子は照合のソート順を使用して評価されます。

次の表は、LIKE、SIMILAR TO、REGEXP が評価した範囲 '[A-C]' に該当する文字を示しています。2 つのデータベースはどちらも 1252LATIN1 照合を使用しますが、最初のデータベースでは大文字と小文字が区別されず、2 番目では区別されます。

	LIKE/SIMILAR TO '[A-C]'	**REGEXP '[A-C]'**
demo.db (大文字と小文字の区別なし)	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 (大文字と小文字を区別)	A、B、C、b、c、À、Á、Â、Ã、Ä、Å、Æ、Ç、ç	A、B、C

この結果から次のことを確認できます。

○LIKE と SIMILAR TO は、範囲内のアクセント記号付き文字を含めます。

○LIKE と SIMILAR TO では、データベースで大文字と小文字が区別されるかどうかに応じて、該当する文字が変わります。具体的には、結果に範囲内の小文字はすべて含まれますが、大文字と小文字が区別されるデータベースでの検索にそのような動作は期待されません。

同様に、大文字と小文字を区別するデータベースで、範囲内に含まれる文字に一貫性がないように見えることがあります。たとえば、大文字と小文字を区別するデータベースでの SIMILAR TO '[A-C]' の結果には、A、b、B、c、C が含まれますが、a は含まれません。これは、ソート順で a が A より前に位置しているためです。

○REGEXP は、データベースで大文字と小文字が区別されるかどうかに関係なく、A、B、C を返します。範囲に小文字も含めるには、その条件を範囲の定義に追加する必要があります。たとえば、REGEXP '[a-cA-C]' のように指定します。

○REGEXP の結果に含まれる文字のセットは、データベースで大文字と小文字が区別されるかどうかに関係なく、同じです。

データベースで使用されている照合、大文字と小文字の区別、アクセント記号の区別が、上記の例と異なる場合でも、データベースに接続して上記の文を実行することによって、同様のテストを実行し、LIKE、SIMILAR TO、REGEXP が返す内容を確認できます。

```
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]';
```

参照

- 「正規表現の概要」17 ページ
- 「正規表現の構文」18 ページ
- 「正規表現の例」31 ページ

LIKE 探索条件

構文

LIKE 探索条件の構文は次のとおりです。

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

パラメーター

- **expression**　検索される文字列。
- **pattern**　*expression* を検索するためのパターン。
- **escape-character**　アンダースコアやパーセント記号などの特殊文字をエスケープするために使用する文字。デフォルトのエスケープ文字は NULL 文字で、文字列リテラル内では '¥x00' のように指定します。

備考

LIKE 探索条件は、*expression* を *pattern* に一致させようとし、TRUE、FALSE、または UNKNOWN と評価されます。

expression が *pattern* に一致すれば、探索条件は TRUE と評価されます (NOT が指定されない場合)。*expression* または *pattern* が NULL 値の場合、探索条件は UNKNOWN と評価されます。NOT キーワードを使用すると探索条件の意味が逆になりますが、UNKNOWN は変わりません。

expression は、CHAR または NCHAR の文字列として解釈されます。*expression* の内容全体が一致に使用されます。同様に、*pattern* は、CHAR または NCHAR の文字列として解釈され、次の表のサポートされているワイルドカードをいくつでも含めることができます。

ワイルドカード	一致するもの
_ (アンダースコア)	任意の 1 文字。たとえば、a_ は ab と ac に一致しますが、a には一致しません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ワイルドカード	一致するもの
% (パーセント記号)	0 個以上の文字からなる任意の文字列。たとえば、bl% は bl と bla に一致します。
[]	指定範囲内、または一連の指定文字の任意の 1 文字。たとえば、T[oi]m は Tom または Tim に一致します。
[^]	指定範囲、または一連の指定文字に**含まれない**任意の 1 文字。たとえば、M[^c] は Mb と Md に一致しますが、Mc には一致しません。

その他すべての文字は正確に一致しなければなりません。

たとえば、次のような探索条件は、文字 a で始まり、末尾から 2 つ目の文字が b の name のローの場合、TRUE になります。

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

escape-character を指定する場合は、シングルバイトの CHAR 文字または NCHAR 文字として評価される文字を指定します。エスケープ文字は、*pattern* 内のパーセント記号、アンダースコア、左側の角カッコ、またはその他のエスケープ文字の前に置くことができます。これは、特殊文字が特別な意味を持たないようにするためです。この方法でエスケープすると、パーセント記号はパーセント記号に一致し、アンダースコアはアンダースコアに一致します。

126 バイト以下のパターンは、すべてサポートされています。ワイルドカードを含まない、126 バイトを超えるパターンはサポートされていません。126 バイトを超えるワイルドカード文字を含むパターンは、パターンの内容によってサポートされることがあります。パターンの表現に使用されるバイト数は、パターンが CHAR か NCHAR かによって異なります。

LIKE 探索条件を使用するさまざまな方法

検索対象	例	その他の情報
一連の文字	LIKE 'sm[iy]th'	検索対象の一連の文字は、角カッコ内にリストして指定します。この例では、探索条件は **smith** と **smyth** に一致します。

検索対象	例	その他の情報
範囲内の文字	LIKE '[a-r]ough'	検索対象の文字範囲は、角カッコ内に範囲を書いて指定します。範囲の始めと終わりの間にハイフンを書きます。この例では、探索条件は bough と rough に一致しますが、tough には一致しません。 文字列の範囲 [a-r] は「a 以上 r 以下」と解釈され、データベースの照合では、> と < 演算子が実行されます。 範囲は、まず小さい方の値、次に大きい方の値を指定してください。たとえば、[z-a] は何も一致しません。これは、[z-a] の範囲にいずれの文字も一致しないためです。
文字範囲と一連の文字の結合	... LIKE '[a-rt]ough'	角カッコ内で、文字範囲と一連の文字を結合できます。この例では、... LIKE '[a-rt]ough' は bough、rough、tough に一致します。 パターン [a-rt] は、a ～ r の範囲に含まれる文字または t と一致する 1 文字と解釈されます。
範囲外の 1 文字	... LIKE '[^a-r]ough'	脱字記号 (^) は、検索から除外する文字範囲を指定します。この例では、LIKE '[^a-r]ough' は文字列 tough に一致しますが、文字列 rough や bough には一致しません。 脱字記号は、この記号以外のカッコ内の内容を否定します。たとえば、角カッコ [^a-rt] は、a ～ r の範囲に含まれない文字かつ t ではない 1 文字と解釈されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

検索対象	例	その他の情報
後続ブランクのある探索パターン	'90 '、'90[ ]'、'90_'	探索パターンに後続ブランクが含まれる場合、データベースサーバーではブランクが含まれる値に対してのみパターンが一致します。文字列へのブランクの埋め込みは行われません。たとえば、検索されている値が幅 3 文字以上の CHAR 型または VARCHAR 型カラムにあっても、'90 '、'90[ ]'、'90_' のパターンは、式 '90 ' には一致しますが、式 '90' には一致しません。

文字範囲と一連の文字の特殊なケース

角カッコ内の任意の 1 文字は、その文字を指しています。たとえば、[a] は、文字 a に一致します。[^] は脱字記号、[%] はパーセント記号 (パーセント記号は、このコンテキストではワイルドカードになりません)、[_] はアンダースコア文字に一致します。また、[[] は文字 [に一致します。

その他の特殊なケースには、次のものがあります。

- パターン [a-] は、文字 a または - に一致します。
- パターン [] は、一致する文字がないのでローを返すことはありません。
- パターン [または [abp-q は、閉じカッコが欠けているため構文エラーを返します。
- 角カッコ内ではワイルドカードは使用できません。パターン [a%b] は a、%、または b のいずれかを検出します。
- 脱字記号を使用しても、カッコ内の先頭になければ範囲を否定できません。パターン [a^b] は a、^、または b のいずれかを検出します。

大文字と小文字の区別と、比較の実行

データベース照合が大文字と小文字を区別する場合は、探索条件でも大文字と小文字を区別します。大文字と小文字を区別する照合で大文字と小文字を区別しない検索を実行するには、大文字と小文字を両方指定します。たとえば、次の探索条件では、文字列 Bough、rough、TOUGH が true と評価されます。

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

文字列単位で比較する等価演算子 (=) やその他の演算子とは異なり、比較が文字単位で実行されます。たとえば、UCA 照合 (照合が UCA に設定された CHAR または NCHAR) で比較が行われる場合、'Æ'='AE' は true ですが、'Æ' LIKE 'AE' は false です。

文字単位で比較する一致では、検索される式に含まれる各文字が 1 文字ごとに一致するか (照合の文字等価を使用します)、または LIKE 式のワイルドカードに一致する必要があります。

各国文字 (NCHAR) サポート

LIKE 探索条件は、CHAR と NCHAR の文字列の比較に使用できます。この場合、共通データ型を使用して比較が行われるように、文字セット変換が実行されます。次に、文字単位の比較が実行されます。

引用符で囲んだ値の前に N を付けることによって (たとえば、expression LIKE N'pattern')、*expression* または *pattern* を NCHAR 文字列リテラルとして指定できます。また、CAST 関数を使用して、CHAR または NCHAR にパターンをキャストすることもできます (たとえば、expression LIKE CAST(pattern AS datatype))。

ブランクが埋め込まれたデータベース

ブランクが埋め込まれたデータベースの場合、*expression* と *pattern* の一致では、左から右の順に文字単位の比較が行われるため、LIKE パターンのセマンティックが変わることはありません。評価時に、*expression* または *pattern* の値に追加のブランクは埋め込まれません。このため、式 a1 はパターン a1 に一致しますが、パターン 'a1 ' (a1 の後ろにスペースがある) または a1_ には一致しません。

標準と互換性

- **SQL/2008**　LIKE 探索条件は、SQL/2008 標準のコア機能です。ただし、SQL Anywhere では大文字と小文字を区別しない照合とブランク埋め込みがサポートされるため、この標準の動作とは微妙な違いがあります。

  SQL Anywhere では、オプションの SQL 言語機能 F281 をサポートしています。これにより、パターンとエスケープ式を実行時に評価される任意の式にできます。機能 F281 を使用すると、*expression* を単純なカラム参照よりも複雑な式にすることもできます。

  角カッコ [] に含まれる文字範囲と一連の文字の使用はベンダー拡張です。

  SQL Anywhere では、SQL/2008 の機能 T042 をサポートしています。これにより、LIKE 探索条件で LONG VARCHAR 値の文字列式を参照できます。

  NCHAR の文字列式またはパターンを指定する LIKE 探索条件は、ANSI SQL/2008 標準のオプションの SQL 言語機能 F421 です。

参照

- 「CHAR と NCHAR の比較」129 ページ
- 「文字列リテラル」7 ページ
- 「CAST 関数 [データ型変換]」165 ページ
- 「LIKE、REGEXP、SIMILAR TO：文字の比較における相違点」40 ページ
- 「LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件」40 ページ
- 「WHERE 句：ローの指定」『SQL Anywhere サーバー SQL の使用法』
- 「REGEXP 探索条件」47 ページ
- 「SIMILAR TO 探索条件」48 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

REGEXP 探索条件

パターンと文字列を一致させます。

構文

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

パラメーター

- **expression**　検索される文字列。
- **pattern**　*expression* を検索するための正規表現。
- **escape-expression**　一致で使用されるエスケープ文字。デフォルトは円記号 (¥) です。

備考

REGEXP 探索条件は文字列全体に一致し、部分文字列には一致しません。部分文字列で文字列に一致させるには、残り部分と一致するワイルドカードで文字列を囲みます (.*pattern.*)。たとえば、SELECT ... WHERE Description REGEXP 'car' は car にのみ一致し、sportscar には一致しません。一方、SELECT ... WHERE Description REGEXP '.*car' は、car や sportscar など、car で終わるすべての文字列に一致します。または、REGEXP_SUBSTR 関数を使用するようにクエリを書き換えることができます。この関数は、文字列内の部分文字列を検索します。

部分文字クラスのみに一致させる場合は、外側の角カッコと、部分文字クラス用の角カッコを使用する必要があります。たとえば expression REGEXP '[[:digit:]]' のように指定します。

データベース照合と一致

REGEXP を使用した場合に、パターンに含まれるリテラルと文字とが一致するのは、厳密な意味で同じ文字の場合 (コードポイント値が同じ場合) のみです。文字クラスによる範囲 (たとえば '[A-F]') と一致する文字は、そのコードポイント値が、範囲の最初に指定されている文字 (A) のコードポイント値以上で、なおかつ、範囲で次に指定されている文字 (F) のコードポイント値以下の場合に限られます。

文字列単位で比較する等価演算子 (=) やその他の演算子とは異なり、比較が文字単位で実行されます。たとえば、UCA 照合 (照合が UCA に設定された CHAR または NCHAR) で比較が行われる場合、'Æ'='AE' は true ですが、'Æ' REGEXP 'AE' は false です。

各国文字 (NCHAR) サポート

REGEXP 探索条件は、CHAR と NCHAR の文字列の比較に使用できます。この場合、共通データ型を使用して比較が行われるように、文字セット変換が実行されます。次に、コードポイント単位の比較が実行されます。

引用符で囲んだ値の前に N を付けることによって (たとえば、expression REGEXP N'pattern')、*expression* または *pattern* を NCHAR 文字列リテラルとして指定できます。また、CAST 関数を使用して、CHAR または NCHAR にパターンをキャストすることもできます (たとえば、expression REGEXP CAST(pattern AS datatype))。

参照

- 「CHAR と NCHAR の比較」129 ページ
- 「文字列リテラル」7 ページ
- 「CAST 関数 [データ型変換]」165 ページ
- 「正規表現の概要」17 ページ
- 「SIMILAR TO 探索条件」48 ページ
- 「LIKE 探索条件」42 ページ
- 「REGEXP_SUBSTR 関数 [文字列]」314 ページ
- 「LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件」40 ページ
- 「正規表現：特殊部分文字クラス」22 ページ

標準と互換性

- **SQL/2008**　REGEXP 探索条件はベンダー拡張ですが、SQL/2008 標準の SQL 言語機能 F841 の LIKE_REGEX 探索条件とほぼ互換性があります。

  SQL Anywhere では、ANSI SQL/2008 の機能 F281 をサポートしています。これにより、パターンとエスケープ式を実行時に評価される任意の式にできます。機能 F281 を使用すると、*expression* を単純なカラム参照よりも複雑な式にすることもできます。

  SQL Anywhere では、ANSI SQL/2008 の機能 T042 をサポートしています。これにより、REGEXP 探索条件で LONG VARCHAR 値の文字列式を参照できます。

  NCHAR の文字列式またはパターンを指定する REGEXP 探索条件は、ANSI SQL/2008 標準の機能 F421 です。

SIMILAR TO 探索条件

パターンと文字列を一致させます。

構文

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

パラメーター

- **expression**　検索される式。

- **pattern**　*expression* を検索するための正規表現。

- **escape-expression**　一致で使用するエスケープ文字。デフォルトのエスケープ文字は NULL 文字で、文字列リテラル内では '¥x00' のように指定します。

正規表現の構文	意味
¥ *x*	*x* と同じものに一致します。ここでは、エスケープ文字は円記号 (¥) と見なされます。たとえば、¥[は '[' に一致します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

正規表現の構文	意味
x	任意の文字 (メタ文字以外) がそれ自体に一致します。たとえば、A は 'A' に一致します。

備考

部分文字列と文字列を一致させるには、パーセント記号のワイルドカードを使用します (%*expression*)。たとえば、SELECT ... WHERE Description SIMILAR TO 'car' は car にのみ一致し、sportscar には一致しません。一方、SELECT ... WHERE Description SIMILAR TO '%car' は、car や sportscar など、car で終わるすべての文字列に一致します。

部分文字クラスのみに一致させる場合は、外側の角カッコと、部分文字クラス用の角カッコを使用する必要があります。たとえば、*expression* SIMILAR TO '[[:digit:]]' のように指定します。

文字列単位で比較する等価演算子 (=) やその他の演算子とは異なり、比較が文字単位で実行されます。たとえば、UCA 照合 (照合が UCA に設定された CHAR または NCHAR) で比較が行われる場合、'Æ'='AE' は true ですが、'Æ' SIMILAR TO 'AE' は false です。

文字単位で比較して一致するには、検索される式に含まれる各文字が、SIMILAR TO パターンに含まれる 1 文字またはワイルドカード文字に一致する必要があります。

データベース照合と一致

SIMILAR TO では、文字が等価かどうかの判断や文字クラス範囲の評価に、照合が使用されます。たとえば、データベースで大文字と小文字やアクセント記号が区別されない場合、一致でも大文字と小文字やアクセント記号が区別されません。範囲も、照合のソート順を使用して評価されます。

各国文字 (NCHAR) サポート

SIMILAR TO 探索条件は、CHAR と NCHAR の文字列の比較に使用できます。この場合、共通データ型を使用して比較が行われるように、文字セット変換が実行されます。次に、文字単位の比較が実行されます。

引用符で囲んだ値の前に N を付けることによって (たとえば、expression SIMILAR TO N'pattern')、*expression* または *pattern* を NCHAR 文字列リテラルとして指定できます。また、CAST 関数を使用して、CHAR または NCHAR にパターンをキャストすることもできます (たとえば、expression SIMILAR TO CAST(pattern AS datatype))。

参照

- 「正規表現の概要」17 ページ
- 「CHAR と NCHAR の比較」129 ページ
- 「文字列リテラル」7 ページ
- 「CAST 関数 [データ型変換]」165 ページ
- 「REGEXP 探索条件」47 ページ
- 「LIKE 探索条件」42 ページ
- 「REGEXP_SUBSTR 関数 [文字列]」314 ページ
- 「正規表現：特殊部分文字クラス」22 ページ
- 「LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件」40 ページ

標準と互換性

- **SQL/2008**　SIMILAR TO 述部は、SQL/2008 標準のオプションの SQL 言語機能 T141 です。

IN 探索条件

構文

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

備考

IN 探索条件は、*query-expression* によって返された値のセットまたは *expression-list* で指定した値のセットと *expression* を比較します。NOT キーワードがない場合、IN 探索条件は次の規則に従って評価されます。

- *expression* が NULL でなく、少なくとも 1 つの値と等しい場合、TRUE です。
- *expression* が NULL で、値リストが空でない場合、または少なくとも 1 つの値が NULL で、*expression* が他の値のいずれとも等しくない場合、UNKNOWN です。
- *expression* が NULL で、*query-expression* が値を返さない場合、または *expression* が NULL でなく、いずれの値も NULL でなく、*expression* がいずれの値とも等しくない場合、FALSE です。

NOT キーワードを指定すると、評価結果の TRUE と FALSE が逆になります。

探索条件 *expression* **IN (** *expression-list* **)** は *expression* = **ANY (** *expression-list* **)** と同義です。

探索条件 *expression* **NOT IN (***expression-list* **)** は *expression* <> **ALL (** *expression-list* **)** と同義です。

expression-list の式には、結果が単一のローと単一のカラムになるリテラル、変数、ホスト変数、またはクエリ式を指定できます。

標準と互換性

- **SQL/2008**　コア機能。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CONTAINS 探索条件

構文

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*

fuzzy-expression :
term
| *fuzzy-expression term*

term :
simple-term
| *prefix-term*

prefix-term :
*simple-term**

phrase :
" *phrase-string* **"**

near-expression :
term **NEAR[***distance***]** *term*
| *term* { **NEAR** | **~** } *term*

phrase-string :
term
| *phrase-string term*

simple-term : 検索に使われる単一のインデックス単語を表す、ホワイトスペースと特殊記号で区切られた文字列。

distance : 正の整数

パラメーター

- **and-expression** *and-expression* は、*primary-expression* と *simple-expression* が両方ともテキストインデックスで見つかる必要があることを指定するために使用します。

  デフォルトでは、単語や式の間に演算子が指定されていない場合は、*and-expression* があるものとして処理されます。たとえば、'a b' は、'a AND b' と解釈されます。

  アンパサンド (&) は AND の代わりとして使用でき、式や単語のどちら側にも配置できます (たとえば、'a &b')。

- **and-not-expression** *and-not-expression* は、*primary-expression* がテキストインデックスに存在し、なおかつ、*basic-expression* がテキストインデックスに**存在してはならない**ことを示すために使用します。これは「否定」とも呼ばれています。

  否定を示すのにハイフンを使用する場合、そのハイフンの左がスペース、右に単語が隣接していることが必要です。それ以外の場合、ハイフンは否定と解釈されません。たとえば、'a - b' は 'a AND NOT b' と同義ですが、'a - b' ではハイフンが無視され、この文字列は 'a AND b' と同義になります。'a-b' はフレーズ '"a b"' と同義です。

- **or-expression** *or-expression* は、*simple-expression* または *contains-query-string* がテキストインデックスに少なくとも 1 つ存在する必要があることを示すために使用します。たとえば、'a|b' は、'a OR b' と解釈されます。

- **fuzzy-expression** *fuzzy-expression* は、指定した単語と似ている単語を検索するために使用します。あいまい一致は、NGRAM テキストインデックスに対してのみサポートされています。

- **near-expression** *near-expression* は、互いの距離が近い単語を検索するために使用します。これは「近接検索」とも呼ばれています。たとえば、'b NEAR[5] c' は、互いの距離が 5 語以下の b と c の出現を探します。単語の順序に意味はなく、'b NEAR c' は 'c NEAR b' と同義です。

  NEAR を *distance* なしで指定すると、デフォルトである 10 語が適用されます。

  NEAR の代わりとしてチルダ (~) を指定することもできます。これは、距離を指定せずに NEAR を指定することと同義で、デフォルトの 10 語が適用されます。チルダを指定した場合、距離は指定できません。常に 10 語になります。

  'a NEAR[1] b NEAR[1] c' のように、NEAR 式を連ねて使用することはできません。

- **prefix-term** *prefix-term* は、指定したプレフィクスで始まる単語を検索するために使用します。たとえば、'datab*' は datab で始まる単語を検索します。これは「プレフィクス検索」とも呼ばれています。プレフィクス検索では、単語のアスタリスクより左の部分に対して一致が実行されます。

備考

CONTAINS 探索条件は、引数としてカラムリストと *contains-query-string* を取ります。これは、探索条件 (述部とも呼ばれる) を指定できる場所ならどこでも使用でき、TRUE または FALSE を返します。*contains-query-string* は、クエリ実行時に認識できる値を持つ、定数文字列または変数である必要があります。*contains-query-string* は、NULL や空の文字列にはできず、300 を超える有効な単語を指定することはできません。有効な単語とは、許可される単語長以内の長さで、STOPLIST に含まれていない単語です。*contains-query-string* に 300 を超える有効な単語がある場合は、エラーが返されます。

テキスト設定のために *contains-query-string* 内のすべての単語が削除される場合、CONTAINS 探索条件の結果は FALSE となります。

contains-query-string の解釈方法の詳細については、「テキスト設定オブジェクトの例」『SQL Anywhere サーバー SQL の使用法』を参照してください。

複数のカラムが指定された場合は、それらがどれも同じベーステーブルを参照している必要があります。テキストインデックスが複数のベーステーブルにわたることはありません。ベーステーブルは FROM 句で直接参照できます。また、ビューまたは派生テーブルで使用することもできます。

クエリ文字列に英数字以外の文字を使用する場合は、次の点に注意が必要です。

- アスタリスクが単語内にあると、エラーが返されます。
- 英数字以外の文字 (特殊文字を含む) を *fuzzy-expression* に使用しないでください。このような文字は、ホワイトスペースとして扱われ、単語区切りとして機能します。
- 可能であれば、特殊文字ではない英数字以外の文字をクエリ文字列に含めないでください。特殊文字ではない英数字以外の文字があると、それを含む単語がフレーズとして扱われ、その文字の位置で単語が区切られます。たとえば、'things we've done' は 'things "we ve" done' と解釈されます。

フレーズの内部にあっても特殊文字として解釈されるのは、アスタリスクのみです。それ以外の特殊文字がフレーズの内部にあると、ホワイトスペースとして扱われ、単語区切りとして機能します。

contains-query-string は、大きく分けて次の 2 ステップで解釈されます。

- **ステップ 1 : 演算子の解釈と優先度の適用**　このステップでは、キーワードが演算子として解釈され、優先度の規則が適用されます。
- **ステップ 2 : テキスト設定オブジェクトの設定の適用**　このステップでは、テキスト設定オブジェクトの設定が単語に適用されます。たとえば、NGRAM テキストインデックスでは、単語は N-gram 表現に分解されます。このステップで、単語長の設定値を超えているクエリ単語またはストップリストに載っているクエリ単語が削除されます。

CONTAINS 探索条件における演算子の優先度

クエリの評価中、式は次の優先順位に従って評価されます。

1. FUZZY、NEAR

2. AND NOT
3. AND
4. OR

キーワードとしての BEFORE の処理

SQL Anywhere では、現在、BEFORE キーワードを演算子としてサポートしていません。たとえば、CONTAINS(column-name, 'a before b') と指定すると、エラーが返されます。代わりに NEAR キーワードを使用してクエリを作成します。

フレーズクエリの一部である場合は、単語 "before" を検索**できます**。たとえば CONTAINS(column-name, '"a before b"') のように指定します。これはフレーズ "a before b" を検索します。

アスタリスク (*) を使用できる構文

アスタリスクは「プレフィクス検索」で使用します。アスタリスクは、クエリ文字列の末尾、または、スペース、アンパサンド、パイプ記号、閉じカッコ、または終了引用符の直前に指定できます。それ以外の位置にアスタリスクを指定すると、エラーが返されます。

次の表は、アスタリスクの使用法として可能なものを示しています。

クエリ文字列	同義の文字列	解釈
'th*'		th で始まる単語を検索します。
'th*&best'	'th* AND best' および 'th* best'	th で始まる単語と、単語 best を検索します。
'th*|best'	'th* OR best'	th で始まる単語、または単語 best を検索します。
'very&(best|th*)'	'very AND (best OR th*)'	単語 very と、単語 best または th で始まる単語を検索します。
'"fast auto*"'		auto で始まる単語が直後に続く、単語 fast を検索します。
'"auto* price"'		単語 price が直後に続く、auto で始まる単語を検索します。

注意
アスタリスクを含むクエリ文字列の解釈は、テキスト設定オブジェクトの設定によって異なります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ハイフン (-) を使用できる構文

ハイフンは、単語や式の「否定」に使用でき、NOT と同義です。ハイフンが否定と解釈されるかどうかは、クエリ文字列内での位置によります。たとえば、ハイフンの直後に単語や式が続く場合、そのハイフンは否定と解釈されます。ハイフンが単語内に埋め込まれている場合は、そのままハイフンとして解釈されます。

否定を示すために使用する場合、ハイフンの直前にホワイトスペース、直後に式がある必要があります。

fuzzy-expression のフレーズ内で使用すると、ハイフンはホワイトスペースとして扱われ、単語区切りとして機能します。

次の表に、ハイフンを使用できる構文を示します。

クエリ文字列	同義の文字列	解釈
'the -best'	'the AND NOT best', 'the AND -best', 'the & -best', 'the NOT best'	単語 the と、best ではない単語を検索します。
'the -(very best)'	'the AND NOT (very AND best)'	単語 the と、very でも best でもない単語を検索します。
'the -"very best"'	'the AND NOT "very best"'	単語 the と、フレーズ very best ではないものを検索します。
'alpha-numerics'	'"alpha numerics"'	単語 numerics が直後に続く、単語 alpha を検索します。
'wild - west'	'wild west'および 'wild AND west'	単語 wild と、単語 west を検索します。

特殊文字を使用できる構文

次の表は、アスタリスクとハイフンを除くすべての特殊文字 を使用できる構文を示します。

以下に示されている特殊文字は、フレーズ内に出現した場合は特殊文字とは見なされず、削除されます。

注意
文字列リテラルの指定に関連する制約事項は、クエリ文字列にも該当します。たとえば、アポストロフィはエスケープする必要があります。

文字または構文	使用例と備考
アンパサンド (&)	アンパサンドは AND と同義で、次のような指定が可能です。 ● 'a & b' ● 'a &b' ● 'a&b' ● 'a& b'
パイプ記号 (|)	パイプ記号は OR と同義で、次のような指定が可能です。 ● 'a|b' ● 'a |b' ● 'a | b' ● 'a| b'
二重引用符 (")	二重引用符は、順序と相対的な距離が重要な単語並びを含める場合に使用します。たとえば、クエリ文字列 'learn "full text search"' において、"full text search" はフレーズです。この例において、learn はこのフレーズの前後どちらにあっても、別のカラムにあってもかまいませんが (テキストインデックスが複数カラムに対して構築されている場合)、完全に一致するフレーズが 1 つのカラム内で見つかる必要があります。
丸カッコ ()	丸カッコは、式の評価順がデフォルトの順序とは違う場合に、その順序を指定するために使用します。たとえば、'a AND (b|c)' は「a と、b または c」と解釈されます。
チルダ (~)	チルダは NEAR[10] と同義です。クエリ文字列 'full~text' は 'full NEAR text' と同義で、「単語 text から 10 語以内の距離にある単語 full」と解釈されます。 距離とチルダを同時に指定することはできません。
角カッコ [ ]	角カッコはキーワード NEAR と組み合わせて *distance* の指定に使用します。角カッコをそれ以外の目的で使用すると、エラーが返されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　CONTAINS 述部はベンダー拡張です。

参照

- 「特殊文字を使用できる構文」55 ページ
- 「近接検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの作成または変更時の指定項目」『SQL Anywhere サーバー SQL の使用法』
- 「ビュー検索」『SQL Anywhere サーバー SQL の使用法』
- 「CONTAINS 探索条件における演算子の優先度」53 ページ
- 「プレフィクス検索」『SQL Anywhere サーバー SQL の使用法』
- 「ハイフン (-) を使用できる構文」55 ページ
- 「アスタリスク (*) を使用できる構文」54 ページ
- 「文字列リテラル」7 ページ
- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの作成または変更時の指定項目」『SQL Anywhere サーバー SQL の使用法』
- 「FROM 句」758 ページ
- 「sa_char_terms システムプロシージャー」1043 ページ
- 「sa_nchar_terms システムプロシージャー」1139 ページ

EXISTS 探索条件

構文

EXISTS (*subquery* **)**

備考

EXISTS 探索条件は、サブクエリ結果にローが少なくとも 1 つあれば TRUE で、ローがなければ FALSE です。EXISTS 探索条件は、UNKNOWN にはなりません。

標準と互換性

- **SQL/2008**　コア機能。

IS NULL および IS NOT NULL 探索条件

構文

expression **IS [NOT] NULL**

備考

NOT キーワードがない場合、expression が NULL 値なら IS NULL 探索条件は TRUE、それ以外の場合は FALSE です。NOT キーワードを使用すると探索条件の意味が逆になります。

標準と互換性

- **SQL/2008** コア機能。

真理値探索条件

構文

IS [**NOT**] *truth-value*

備考

NOT キーワードがない場合、*condition* が指定された *truth-value* (TRUE、FALSE、UNKNOWN のいずれか) と評価されれば、探索条件は TRUE になります。それ以外の場合、値は FALSE です。NOT キーワードを使用すると探索条件の意味が逆になりますが、UNKNOWN は変わりません。

標準と互換性

- **SQL/2008** 真理値探索条件は、SQL/2008 標準のオプションの SQL 言語機能 F571 によって構成されています。

トリガーオペレーション条件

構文

trigger-operation:
INSERTING
| **DELETING**
| **UPDATING** [**(** *column-name-string* **)**]
| **UPDATE (** *column-name* **)**

備考

トリガーオペレーション条件は、トリガーのみで使用でき、トリガーを起動させたアクションの種類に応じてそれぞれ異なったアクションを実行するために使用します。

UPDATING の引数は、引用符付きの文字列です (たとえば、UPDATING('mycolumn'))。UPDATE の引数は、識別子です (たとえば、UPDATE(mycolumn))。この 2 つのバージョンは相互運用可能で、他のベンダが提供する DBMS の SQL ダイアレクトとの互換性のために用意されています。

UPDATING または UPDATE 関数を指定する場合は、CREATE TRIGGER 文で REFERENCING 句を指定して、構文エラーを回避してください。

例

次のトリガーでは、トリガーが起動されるきっかけとなったアクションを示すメッセージが、Interactive SQL の **[結果]** ウィンドウ枠の **[メッセージ]** タブに表示されます。

```
CREATE TRIGGER tr BEFORE INSERT, UPDATE, DELETE
ON sample_table
REFERENCING OLD AS t1old
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
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;
```

参照

- 「BEGIN 文」497 ページ
- 「CREATE TRIGGER 文」670 ページ
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

3 値的論理

次の表は、3 値的論理で SQL の論理演算子 AND、OR、NOT、IS がどのように機能するかを示します。

AND 演算子

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

OR 演算子

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

NOT 演算子

TRUE	FALSE	UNKNOWN
FALSE	TRUE	UNKNOWN

IS 演算子

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

標準と互換性

- **SQL/2008**　コア機能。真理値テスト (IS UNKNOWN など) は、SQL 言語機能 F571 によって構成されています。

参照

- 「NULL 特別値」67 ページ

明示的な選択性推定

SQL Anywhere は、統計情報を基に各文の実行に最も効率的な方法を判別します。SQL Anywhere は、それらの統計を自動的に収集、更新します。これらの統計は、データベースのシステムテーブル ISYSCOLSTAT に永久的に格納されます。ある文の処理中に収集された統計は、以降の文の効率的な実行方法を見いだすときに使用できます。

場合によっては、統計情報が不正確になったり、関連統計情報が使用不能になったりすることがあります。このような状況がもっとも発生しやすいのは、大量のデータが追加、更新、または削除されてから実行されたクエリが少ない場合です。このような場合は、CREATE STATISTICS 文を実行します。

特定の実行プランに問題がある場合は、オプティマイザーに関するヒントを使用して、特定のインデックスの使用を要求できます。

ただし、特異な状況では、この方法では効果的でないことがあります。そのような場合、明示的な選択性推定を指定することでパフォーマンスを改善できることがあります。

オプティマイザーは対象となる各テーブルについて、結果の一部となるローの数を推測します。条件の成功する確率がオプティマイザーの推定とは異なることがあらかじめ判明している場合、ユーザー推定を明示的に探索条件として指定できます。

予測値はパーセントで表します。この値は、正の整数または小数です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

警告
進行ベースで使用される文には、できるかぎり明示的な推定を指定しないようにしてください。データが変更されると、明示的な推定が不正確になり、オプティマイザーが誤って不適切なプランを選択することがあります。明示的な選択性推定を使用する場合は、数値が正確であることを確認してください。たとえば、0% または 100% の値を指定してインデックスを強制的に使用することはしないでください。

ユーザー推定を無効にするには、データベースオプション user_estimates を Off に設定します。user_estimates のデフォルト値は Override-Magic です。これは、オプティマイザーが条件に MAGIC (デフォルト) 選択性値を使用する場合にのみ、ユーザー提供の選択性推定が使用されることを意味します。オプティマイザーは、述部の選択性を正確に予測できない場合の最終手段として MAGIC 値を使用します。

例

次のクエリでは、ShipDate 値の 1% が 2001/06/30 より遅くなる予測値を出力します。

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

次のクエリでは、ローの 0.5% が条件を満たす予測値を出力します。

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

小数を使用すると、ジョインや大きなテーブルのユーザー予測値はさらに正確になります。

標準と互換性

- **SQL/2008**　ベンダー拡張。

参照

- 「CREATE STATISTICS 文」643 ページ
- 「FROM 句」758 ページ
- 「user_estimates オプション」『SQL Anywhere サーバー データベース管理』
- 「オプティマイザーの推定と統計」『SQL Anywhere サーバー SQL の使用法』

特別値

特別値は、式や、テーブル作成時のカラムのデフォルトに使用できます。

クエリに使用できる特別値もありますが、カラムのデフォルト値だけに使用できる特別値もあります。たとえば、**LAST USER**、**TIMESTAMP**、**UTC TIMESTAMP** は、デフォルト値だけに使用できます。

CURRENT DATABASE 特別値

CURRENT DATABASE は、現在のデータベースの名前を返します。

データ型

文字列

参照

- 「式」12 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

CURRENT DATE 特別値

CURRENT DATE は、現在の年、月、日を返します。

データ型

DATE

参照

- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。ANSI SQL/2008 標準では、現在の日付を定義する特別登録を CURRENT_DATE と呼びます。SQL Anywhere では、CURRENT_DATE をサポートしていません。

CURRENT PUBLISHER 特別値

CURRENT PUBLISHER は、SQL Remote レプリケーション用データベースのパブリッシャーユーザー ID を含む文字列を返します。

データ型

文字列

備考

CURRENT PUBLISHER は文字データ型のカラムでデフォルト値として使用できます。

参照

- 「GRANT PUBLISH 文 [SQL Remote]」 777 ページ
- 「REVOKE PUBLISH 文 [SQL Remote]」 885 ページ
- 「パブリッシャーの表示」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

CURRENT REMOTE USER 特別値

現在の接続が SQL Remote の受信フェーズに属している場合、CURRENT REMOTE USER は、現在、この接続に適用されているメッセージを作成したリモートユーザーのユーザー ID を返します。それ以外のすべての状況では、CURRENT REMOTE USER は NULL 値です。

データ型

文字列

備考

CURRENT REMOTE USER 特別値は、メッセージをデータベースに適用するときに、SQL Remote の受信フェーズによって設定されます。CURRENT REMOTE USER 特別値は、適用される操作が SQL Remote の受信フェーズによって適用されるかどうか、および、そのことが当てはまる場合に、適用される操作をどのリモートユーザーが生成したかをトリガーで判別する場合に、非常に役立ちます。

参照

- 「CURRENT REMOTE USER 特別値の使用」『SQL Remote』
- -t オプション、SQL Remote Message Agent ユーティリティ (dbremote)『SQL Remote』

標準と互換性

ベンダー拡張。

CURRENT TIME 特別値

CURRENT TIME は現在の時、分、秒 (小数位あり) を返します。

データ型

TIME

備考

秒は小数第 6 位まで格納されます。現在時刻の精度はシステムクロックの精度によって制限されます。

参照

- 「式」12 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。ANSI SQL/2008 標準では、現在の時刻を定義する特別登録を CURRENT_TIME と呼びます。SQL Anywhere では、CURRENT_TIME はサポートされていません。

CURRENT TIMESTAMP 特別値

CURRENT TIMESTAMP は、CURRENT DATE と CURRENT TIME を結合して、TIMESTAMP 値を形成します。年、月、日、時、分、秒、秒の小数位で構成されます。

データ型

TIMESTAMP

備考

秒は小数第 6 位まで格納されます。現在時刻の精度はシステムクロックの精度によって制限されます。

DEFAULT TIMESTAMP とは異なり、DEFAULT CURRENT TIMESTAMP で宣言されたカラムにユニークな値を指定する必要はありません。ユニークな値にする必要がある場合、代わりに DEFAULT TIMESTAMP を使用してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CURRENT TIMESTAMP が返す情報は、GETDATE 関数と NOW 関数が返す情報と同じです。

CURRENT_TIMESTAMP は、CURRENT TIMESTAMP と同じです。

注意
DEFAULT CURRENT TIMESTAMP と DEFAULT TIMESTAMP の主な違いは、DEFAULT CURRENT TIMESTAMP は INSERT にのみ設定され、DEFAULT TIMESTAMP は INSERT と UPDATE の両方に設定されることです。

参照

- 「式」12 ページ
- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。SQL/2008 標準では、現在のタイムスタンプを定義する特別登録を CURRENT_TIMESTAMP と呼びます。

CURRENT USER 特別値

CURRENT USER には、現在の接続のユーザー ID が含まれます。

データ型

文字列

備考

CURRENT USER は文字データ型のカラムでデフォルト値として使用できます。

UPDATE 時には、明示的に更新されないかぎり CURRENT USER のデフォルトが入ったカラムは変更されません。LAST USER のデフォルトは、ユーザーによる更新の追跡に使用されることがあります。

CURRENT_USER は、CURRENT USER と同じです。

参照

- 「式」12 ページ
- 「LAST USER 特別値」67 ページ
- 「USER 特別値」74 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。SQL/2008 標準では、現在のユーザーを定義する特別登録を CURRENT_USER と呼びます。

CURRENT UTC TIMESTAMP 特別値

CURRENT UTC TIMESTAMP は、年、月、日、時、分、秒、秒の小数位、タイムゾーンで構成される協定世界時 (UTC: Coordinated Universal Time) を返します。

データ型

TIMESTAMP WITH TIME ZONE

備考

この機能により、データが入力されたタイムゾーンに関係なく、入力データに一貫した時間基準を適用することができます。

参照

- 「式」12 ページ
- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。TIMESTAMP WITH TIME ZONE データ型は、SQL/2008 標準のオプションの SQL 言語機能 F411 です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

LAST USER 特別値

LAST USER は、ローを最後に変更したユーザーのユーザー ID です。

データ型

文字列

備考

LAST USER は文字データ型のカラムでデフォルト値として使用できます。

INSERT の場合、この定数は CURRENT USER と同じ効果があります。

UPDATE では、LAST USER のデフォルト値を持つカラムが明示的に変更されていなければ、現在のユーザー名に変更されます。

DEFAULT TIMESTAMP と結合すると、LAST USER のデフォルト値を使用して、ローを最後に変更したユーザーと日時の両方を記録できます (ただし、別々のカラムに記録されます)。

参照

- 「CREATE TABLE 文」651 ページ
- 「CURRENT USER 特別値」65 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「USER 特別値」74 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

NULL 特別値

NULL 値は、不定または該当なしの値を指定します。

構文

NULL

備考

NULL は、あらゆるデータ型の有効な値とは異なる特別値です。ただし、NULL 値はすべてのデータ型で使用できます。NULL は、情報が不足または不適切であることを表すために使用します。次に示すとおり、NULL が使用される 2 つのケースは、独立していて、性質が異なります。

状況	説明
不足	フィールドに値はありますが、その値が不明です。

状況	説明
不適切	フィールドは、この特定のローには適していません。

SQL では、NOT NULL 制限を使用してカラムを作成できます。このカラムには NULL 値を挿入できません。

NULL 値によって、SQL に 3 値的論理の概念が導入されました。NULL 値をどのような値 (NULL 値を含む) や比較演算子を使用して比較しても、結果は "UNKNOWN" です。この場合、TRUE を返す唯一の探索条件は IS NULL 述部です。SQL では、WHERE 句の探索条件が TRUE と評価された場合のみ、ローが選択されます。UNKNOWN または FALSE と評価されたローは、選択されません。

NULL 値に対するカラム領域の使用量は、カラム当たり 1 ビットであり、領域は 8 ビット単位で割り付けられます。NULL のビット使用量は、NULL 値を許容するテーブルのカラム数に基づいて決められます。

IS [NOT] *truth-value* 句は、NULL 値があるローを選択するために使用します (*truth-value* は TRUE、FALSE、または UNKNOWN のいずれかです)。

カラム Salary に NULL が格納されている場合の例を次に示します。

条件	真理値	選択
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

2 つの異なるテーブルのカラムを比較する場合、同じ規則が適用されます。そのため、2 つのテーブルを結合すると、比較したカラムに NULL 値があるローは選択されません。

数値式で使用する場合も、NULL 値は特別な性質を持っています。NULL 値が含まれる数値式の結果は、すべて NULL 値になります。NULL 値を数値に加算しても結果は NULL 値であり、数値にはなりません。NULL 値を 0 として扱う場合は、**ISNULL(** *expression*, **0)** 関数を使用します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL クエリの作成で生じるエラーの多くは、NULL の性質によるものです。注意して、このような問題を避けるようにしてください。

セット演算子と DISTINCT 句

SQL では、探索条件内での NULL との比較の結果は UNKNOWN となります。ただし、2 つのローがそれぞれの重複であるかどうかを判別する場合、SQL は NULL を NULL と等しいものとして処理します。これらのセマンティックは、集合演算子 (UNION、INTERSECT、EXCEPT)、GROUP BY、WINDOW 句内 PARTITION、および SELECT DISTINCT に適用されます。

たとえば、テーブル T1 で、redundant というカラムのすべてのローに NULL が含まれる場合、次の文は 1 つのローを返します。

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

パーミッション

データベースに接続されている必要があります。

関連する動作

なし。

標準と互換性

- **SQL/2008**　コア機能。
- **Transact-SQL**　Adaptive Server Enterprise では、一部のコンテキストにおいて NULL 値との比較の処理が異なります。*expression* が等号または不等号を使用して変数または NULL リテラルと比較され、*expression* がベーステーブルまたはビューのカラムを参照する単純な式である場合、NULL = NULL が UNKNOWN ではなく TRUE を返す 2 値的論理を使用して比較が実行されます。これらのセマンティックで考えられる比較とそれに相当する SQL/2008 の比較のリストを以下に示します。

Transact-SQL の比較	SQL/2008 の対応する比較
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 では、ansinull オプションが OFF に設定されている場合、これらのセマンティックを実装して、Adaptive Server Enterprise の動作と一致させます。Open Client 接続と jConnect 接続では、ansinull オプションはデフォルトで OFF に設定されます。SQL/2008 のセマンティックを保証するには、ansinull オプションを ON にリセットするか、等号比較の代わりに IS [NOT] NULL 述部を使用します。

SQL Anywhere のユニークインデックスは、エントリに NULL があってもよい点を除き、Adaptive Server Enterprise の場合と同じです。Adaptive Server Enterprise では、そのようなエントリをユニークインデックスに入れることはできません。

jConnect を使用する場合、tds_empty_string_is_null オプションは、空文字列が NULL 文字列または 1 つのブランク文字を含む文字列として返されるかどうかを制御できます。

参照

- 「tds_empty_string_is_null オプション」『SQL Anywhere サーバー データベース管理』
- 「式」12 ページ
- 「ansinull オプション」『SQL Anywhere サーバー データベース管理』
- 「tds_empty_string_is_null オプション」『SQL Anywhere サーバー データベース管理』
- 「探索条件」34 ページ

例

次の INSERT 文は、Borrowed_book テーブルの date_returned カラムに NULL を挿入します。

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

SQLCODE 特別値

SQLCODE は、最後に実行された SQL 文の処理を示します。

データ型

符号付き INTEGER

備考

データベースサーバーは、実行する各 SQL 文に対して、SQLSTATE と SQLCODE を設定します。SQLCODE は製品固有であり (たとえば、Mobile Link には固有の SQLCODE があります)、SQLSTATE についての追加情報を得る場合に使用できます。たとえば、100 以外の正の値は、製品固有の*警告*条件を示します。負の値は、製品固有の*例外*条件を示します。値 100 は、(たとえば、カーソルによってフェッチされた結果セットの最後で)「データがない」ことを示します。

SQLSTATE と SQLCODE は、各 SQLCODE が SQLSTATE に対応しており、各 SQLSTATE が 1 つ以上の SQLCODE に対応していることがあるという点で関連しています。

SQLCODE に関連付けられたエラー条件を返すには、ERRORMSG 関数を使用できます。

注意
SQLSTATE は、SQL 文の結果の優先ステータスインジケーターです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SQLSTATE 特別値」71 ページ
- 「SQL Anywhere のエラーメッセージ (SQLCODE 順)」『エラーメッセージ』
- 「式」12 ページ
- 「ERRORMSG 関数 [その他]」218 ページ

標準と互換性

- **SQL/2008**　SQLCODE は、ANSI SQL/1992 標準で非推奨になり、SQL/1999 から完全に削除されました。SQL Anywhere では、アプリケーションの下位互換のため、SQLCODE 値を引き続き維持しています。SQLSTATE は、優先ステータスインジケーターです。

SQLSTATE 特別値

SQLSTATE は、最後に実行された SQL 文が成功、エラー、または警告条件になったかどうかを示します。

データ型

文字列

備考

データベースサーバーは、実行する各 SQL 文に対して、SQLSTATE と SQLCODE を設定します。SQLSTATE は、最後に実行された SQL 文が成功、警告、またはエラー条件になったかどうかを示す文字列です。

各 SQLSTATE は、すべてのプラットフォームに共通するエラーを表し、通常は製品共通の表現が含まれてます。SQLSTATE 値のフォーマットは、2 文字クラス値の後ろに 3 文字サブクラス値が続きます。クラス値とサブクラス値に関する SQLSTATE 適合のガイドラインは、ISO/ANSI SQL 標準に概説されています。

SQL Anywhere は、ISO/ANSI の SQLSTATE 規則に準拠しており、次の追加項目と例外項目があります。

クラスとサブクラス	条件
01WC*x*	文字セット変換に関する警告
38*xxx*	外部関数の例外
42X*xx*	構文エラー：式
42R*xx*	構文エラー：参照整合性 (たとえば、2 つ目のプライマリキーを作成しようとした場合)
42W*xx*	構文エラー：汎用

クラスとサブクラス	条件
42U*xx*	構文エラー：重複したオブジェクト参照、定義されていないオブジェクト参照、またはあいまいなオブジェクト参照
42Z*xx*	アクセス違反
54W*xx*	製品の制限超過
55W*xx*	オブジェクトが操作を成功させるために必要なステータスにない
57*xxx*	リソース使用不可またはオペレーター介入
5R*xxx*	SQL Remote のエラー
WB*xxx*	オンラインバックアップのエラー
WI*xxx*	データベースの内部エラー
WP*xxx*	プロシージャー、変数などでのエラー
WL*xxx*	ロードかアンロードまたはその両方のエラー
WW*xxx*	SQL Anywhere 固有のその他のエラー／警告 (システム障害を含む)
WO*xxx*	リモートデータアクセス機能に関するエラー
WJ*xxx*	JCS と JDBC 関連のエラー
WC*xxx*	文字変換のエラー
WX*xxx*	XML 関連のエラー
WT*xxx*	テスト関連のエラー

正常終了クラスは '00*xxx*' です (たとえば、'00000')。

SQLSTATE と SQLCODE は、各 SQLCODE が SQLSTATE に対応しており、各 SQLSTATE が 1 つ以上の SQLCODE に対応していることがあるという点で関連しています。

SQLSTATE に関連付けられたエラー条件を返すには、ERRORMSG 関数を使用できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「ERRORMSG 関数 [その他]」218 ページ
- 「SQLCODE 特別値」70 ページ
- 「SQL Anywhere のエラーメッセージ (SQLSTATE 順)」『エラーメッセージ』
- 「式」12 ページ

標準と互換性

- **SQL/2008**　値 '0' ～ '4' と 'A' ～ 'H' で始まる SQLSTATE クラス (最初の 2 文字) は、ANSI 標準で定義されています。その他のクラスは実装依存です。同様に、値 '0' ～ '4' と 'A' ～ 'H' で始まるサブクラス値も、ANSI 標準で定義されています。これらの範囲に含まれないサブクラス値は実装依存です。

TIMESTAMP 特別値

TIMESTAMP デフォルト値は、テーブル内のローが最後に変更されたローカル日時の記録に使用します。

データ型

TIMESTAMP

備考

秒は小数第 6 位まで格納されます。現在時刻の精度はシステムクロックの精度によって制限されます。

カラムの宣言に DEFAULT TIMESTAMP が指定されている場合は、ローを挿入するとタイムスタンプのデフォルト値が割り付けられます。この値は、ローが更新されるたびに現在の日付と時刻に基づいて更新されます。

DEFAULT TIMESTAMP で宣言されたカラムにはユニークな値が入ります。これにより、アプリケーションは、ほぼ同時に行われた同じローの更新を検出できます。現在のタイムスタンプ値が直前の値と同じ場合は、default_timestamp_increment オプションの値が加えられます。

default_timestamp_increment オプションに基づいて、SQL Anywhere のタイムスタンプ値を自動的にトランケートできます。これは、記録されるタイムスタンプ値の精度が低い他のデータベースソフトウェアとの互換性を維持する場合に便利です。

グローバル変数 @@dbts は、DEFAULT TIMESTAMP を使用するカラムの最後に生成された値を表す TIMESTAMP 値を返します。

注意
DEFAULT CURRENT TIMESTAMP と DEFAULT TIMESTAMP の主な違いは、DEFAULT CURRENT TIMESTAMP は INSERT にのみ設定され、DEFAULT TIMESTAMP は INSERT と UPDATE の両方に設定されることです。

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「default_timestamp_increment オプション」『SQL Anywhere サーバー データベース管理』
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「truncate_timestamp_values オプション」『SQL Anywhere サーバー データベース管理』
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

USER 特別値

USER には、現在の接続のユーザー ID が含まれます。

データ型

文字列

備考

USER は文字データ型のカラムでデフォルト値として使用できます。これは、CURRENT USER と同義です。

UPDATE 時には、明示的に更新されないかぎり USER のデフォルトが入ったカラムは変更されません。代わりに、LAST USER のデフォルトがユーザーによる更新の追跡に使用されることがあります。

参照

- 「式」12 ページ
- 「CURRENT USER 特別値」65 ページ
- 「LAST USER 特別値」67 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

UTC TIMESTAMP 特別値

UTC TIMESTAMP デフォルト値は、テーブル内のローが最後に変更された協定世界時 (UTC: Coordinated Universal Time) の記録に使用します。

データ型

TIMESTAMP WITH TIME ZONE

備考

秒は小数第 6 位まで格納されます。現在時刻の精度はシステムクロックの精度によって制限されます。

カラムの宣言に DEFAULT UTC TIMESTAMP が指定されている場合は、ローを挿入するとタイムスタンプのデフォルト値が割り付けられます。この値は、ローが更新されるたびに現在の UTC の日付と時刻に基づいて更新されます。

DEFAULT UTC TIMESTAMP で宣言されたカラムにはユニークな値が入ります。これにより、アプリケーションは、ほぼ同時に行われた同じローの更新を検出できます。現在の UTC タイムスタンプ値が直前の値と同じ場合は、default_timestamp_increment オプションの値が加えられます。

default_timestamp_increment オプションを使用して、SQL Anywhere の UTC TIMESTAMP 値を自動的にトランケートできます。これは、記録されるタイムスタンプ値の精度が低い他のデータベースソフトウェアとの互換性を維持する場合に便利です。

注意
DEFAULT UTC TIMESTAMP は INSERT と UPDATE の両方で設定され、DEFAULT CURRENT UTC TIMESTAMP は INSERT で設定されます。

参照

- 「式」12 ページ
- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「default_timestamp_increment オプション」『SQL Anywhere サーバー データベース管理』
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「truncate_timestamp_values オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

変数

SQL Anywhere では、3 つのレベルの変数をサポートしています。

- **ローカル変数**　DECLARE 文を使用して、プロシージャーまたはバッチ内の複合文で定義します。これらは、複合文内にのみ指定できます。

- **接続レベル変数**　CREATE VARIABLE 文で定義します。現在の接続に属し、データベース接続を切断するか、または DROP VARIABLE 文を使用すると消去されます。

- **グローバル変数**　システム定義の値を保有する、システム定義の変数。グローバル変数の名前は、すべて 2 つのアットマーク (@) で始まります。たとえば、グローバル変数 **@@version** の値は、データベースサーバーの現在のバージョン番号です。ユーザーはグローバル変数を定義できません。

ローカル変数と接続レベル変数は、ユーザーが宣言します。これらは、プロシージャーや SQL 文のバッチで使用され、情報を保持します。グローバル変数は、システム定義の値を指定したシステム定義の変数です。

参照

- 「TIMESTAMP データ型」118 ページ
- 「CREATE VARIABLE 文」679 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** SQL ストアドプロシージャーまたはファンクション内で DECLARE 文を使用して宣言される変数は、ANSI SQL/2008 標準では SQL 言語機能 P002 の "Computational completeness" としてサポートされています。CREATE VARIABLE、DROP VARIABLE、およびグローバル変数はすべてベンダー拡張です。

ローカル変数

ローカル変数は DECLARE 文で宣言し、複合文 (BEGIN キーワードと END キーワードで囲まれたブロック) の中でのみ使用できます。SQL Anywhere では、各 DECLARE 文で宣言できる変数は 1 つのみです。

DECLARE が複合文で実行される場合、スコープは複合文に制限されます。

変数の初期設定値は NULL です。変数の値は、SET 文で設定するか、INTO 句のある SELECT 文で割り当てることができます。

DECLARE 文の構文は、次のとおりです。

```
DECLARE variable-name data-type
```

ローカル変数は、プロシージャーが複合文中から呼び出されるかぎり、プロシージャーへ引数として引き渡すことができます。

例

次のバッチは、ローカル変数の使用例を示します。

```
BEGIN
  DECLARE local_var INT;
  SET local_var = 10;
  MESSAGE 'local_var = ', local_var TO CLIENT;
END
```

Interactive SQL からこのバッチを実行すると、local_var = 10 というメッセージが Interactive SQL の **[結果]** ウィンドウ枠の **[メッセージ]** タブに表示されます。

変数 local_var は、変数が宣言された複合文の外側には存在しません。次のバッチは無効で、エラーになります。

```
-- This batch is invalid.
BEGIN
  DECLARE local_var INT;
  SET local_var = 10;
END;
MESSAGE 'local_var = ', local_var TO CLIENT;
```

次の例は、INTO 句のある SELECT 文を使用してローカル変数を設定する方法を示します。

```
BEGIN
  DECLARE local_var INT;
  SELECT 10 INTO local_var;
```

```
  MESSAGE 'local_var = ', local_var TO CLIENT;
END
```

Interactive SQL からこのバッチを実行すると、local_var = 10 というメッセージが Interactive SQL の **[結果]** ウィンドウ枠の **[メッセージ]** タブに表示されます。

標準と互換性

- **SQL/2008**　DECLARE 文は、ANSI SQL/2008 標準では SQL 言語機能 P002 の "Computational completeness" としてサポートされています。

参照

- 「カラム '%1' が見つかりません。」『エラーメッセージ』
- 「Transact-SQL プロシージャーの中の変数」『SQL Anywhere サーバー SQL の使用法』

接続レベル変数

接続レベル変数は、CREATE VARIABLE 文で宣言します。接続レベル変数は、プロシージャーにパラメーターとして引き渡すことができます。

CREATE VARIABLE 文の構文は、次のとおりです。

CREATE VARIABLE *variable-name data-type*

変数が作成されると、値は NULL に初期設定されます。SET 文または INTO 句のある SELECT 文を使用すると、接続レベル変数はローカル変数と同じ方法で設定できます。

接続レベル変数は、接続が終了するまで、または DROP VARIABLE 文を使用して変数が明示的に削除されるまで存在します。次の文は、変数 con_var を削除します。

```
DROP VARIABLE con_var;
```

例

次の SQL 文のバッチは、接続レベル変数の使用例を示します。

```
CREATE VARIABLE con_var INT;
SET con_var = 10;
MESSAGE 'con_var = ', con_var TO CLIENT;
```

Interactive SQL からこのバッチを実行すると、con_var = 10 というメッセージが Interactive SQL の **[結果]** ウィンドウ枠の **[メッセージ]** タブに表示されます。

標準と互換性

- **SQL/2008**　ベンダー拡張。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

グローバル変数

グローバル変数は、データベースサーバーで設定された値を持っています。たとえば、グローバル変数 @@version の値は、データベースサーバーの現在のバージョン番号です。

グローバル変数は、名前の先頭に付けられた 2 つのアットマーク (@) で、ローカル変数や接続レベル変数と区別されます。たとえば、@@error や @@rowcount はグローバル変数です。ユーザーはグローバル変数を定義できません。また、グローバル変数の値は直接更新できません。

一部のグローバル変数 (@@identity など) は、接続に固有の情報と値を保持します。その他の変数 (@@connections など) は、すべての接続に共通の値を保持します。

グローバル変数と特別値

特別値 (CURRENT DATE、CURRENT TIME、USER、SQLSTATE など) は、グローバル変数に類似しています。

次の文は、バージョングローバル変数を呼び出します。

```
SELECT @@version;
```

プロシージャーとトリガーでは、グローバル変数は変数リストから選択できます。次のプロシージャーでは、*ver* パラメーター内にあるサーバーのバージョン番号を返します。

```
CREATE PROCEDURE VersionProc ( OUT ver VARCHAR(100) )
 BEGIN
   SELECT @@version
   INTO ver;
 END;
```

Embedded SQL の場合、グローバル変数はホスト変数リストから選択できます。

サポートされているグローバル変数

変数名	意味
@@char_convert	0 (Transact-SQL との互換性を保つための値)。
@@client_csid	-1 (Transact-SQL との互換性を保つための値)。
@@client_csname	NULL (Transact-SQL との互換性を保つための値)。
@@connections	最後のサーバー起動後に行われたログインの数。
@@cpu_busy	0 (Transact-SQL との互換性を保つための値)。

変数名	意味
@@dbts	DEFAULT TIMESTAMP で定義されたすべてのカラムの最後に生成された値を表す TIMESTAMP 値。
@@error	直前に実行された文の成功または失敗をチェックする Transact-SQL エラーコード。直前のトランザクションが成功していた場合は、0 が返されます。直前のトランザクションが失敗だった場合は、システムによって生成された最後のエラー番号が返されます。@@error が返す値の説明については、「Transact-SQL プロシージャーでのエラー処理」『SQL Anywhere サーバー SQL の使用法』を参照してください。 if @@error != 0 return のような文を指定すると、エラー発生時に処理が終了されます。PRINT 文や IF テストを含めて、すべての SQL 文は @@error をリセットするため、エラーのステータスチェックは、成否を確認する対象の文の直後に行う必要があります。
@@fetch_status	最後のフェッチ文によって得られたステータス情報を保持します。この機能は @@sqlstatus と同じですが、@@sqlstatus とは異なる値を返します。Microsoft SQL Server との互換性のために用意されています。@@fetch_status には、次のいずれかの値が格納されます。 ●0：フェッチ文は正常終了した。 ●-1：フェッチ文はエラーになった。 ●-2：結果セットにこれ以上データがない。
@@identity	INSERT 文または SELECT INTO 文によって IDENTITY カラムまたは DEFAULT AUTOINCREMENT カラムに挿入された最後の値。
@@idle	0 (Transact-SQL との互換性を保つための値)。
@@io_busy	0 (Transact-SQL との互換性を保つための値)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

変数名	意味
@@isolation	接続の現在の独立性レベル。@@isolation は、アクティブレベルの値を取ります。
@@langid	現在の接続で使用中の言語のユニークな言語 ID。
@@language	接続で使用されている言語の名前を返します。
@@max_connections	パーソナルサーバーの場合、サーバーに対して確立できる同時接続最大数 (10 接続)。ネットワークサーバーの場合、アクティブクライアントの最大数 (各クライアントが複数の接続をサポートできるため、データベース接続とは異なります)。
@@maxcharlen	CHAR 文字セットの文字の最大長 (バイト単位)。
@@ncharsize	NCHAR 文字セットの文字の最大長 (バイト単位)。
@@nestlevel	-1 (Transact-SQL との互換性を保つための値)。
@@pack_received	0 (Transact-SQL との互換性を保つための値)。
@@pack_sent	0 (Transact-SQL との互換性を保つための値)。
@@packet_errors	0 (Transact-SQL との互換性を保つための値)。
@@procid	現在実行されているプロシージャーのストアドプロシージャー ID。

変数名	意味
@@rowcount	最後の文の影響を受けるローの数。 @@rowcount の値は、文の直後にチェックする必要があります。 INSERT、UPDATE、DELETE の各文は、影響を受けたローの数を @@rowcount に設定します。 カーソルを使用する場合、@@rowcount は、最後のフェッチ要求までに、カーソルの結果セットからクライアントに返されたローの累積数を表します。 IF 文のようにローに影響を及ぼさない文によって、@@rowcount が 0 にリセットされることはありません。
@@servername	現在のデータベースサーバーの名前。
@@spid	現在の接続の接続ハンドル。これは、sa_conn_info プロシージャーによって表示されるものと同じ値です。
@@sqlstatus	最後のフェッチ文によって得られたステータス情報を保持します。@@sqlstatus には、次のいずれかの値が格納されます。 ●0：フェッチ文は正常終了した。 ●1：フェッチ文はエラーになった。 ●2：結果セットにこれ以上データがない。
@@textsize	SET TEXTSIZE オプションの現在の値。これは、SELECT 文で返される TEXT または IMAGE データのバイト単位の最大長です。デフォルト設定は 32765 (READTEXT を使用して返すことができる最長のバイト文字列)。この値は、SET 文を使用して設定できます。
@@thresh_hysteresis	0 (Transact-SQL との互換性を保つための値)。
@@timeticks	0 (Transact-SQL との互換性を保つための値)。
@@total_errors	0 (Transact-SQL との互換性を保つための値)。
@@total_read	0 (Transact-SQL との互換性を保つための値)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

変数名	意味
@@total_write	0 (Transact-SQL との互換性を保つための値)。
@@tranchained	現在のトランザクションモード。非連鎖の場合は 0、連鎖の場合は 1 です。
@@trancount	トランザクションのネストレベル。バッチ内の BEGIN TRANSACTION ごとに、トランザクションカウントが増分されます。
@@transtate	-1 (Transact-SQL との互換性を保つための値)。
@@version	SQL Anywhere の現在のバージョン番号。

参照

- 「@@identity グローバル変数」83 ページ

@@identity グローバル変数

@@identity 変数には、現在の接続によって IDENTITY カラム、DEFAULT AUTOINCREMENT カラム、または DEFAULT GLOBAL AUTOINCREMENT カラムに挿入された最新の値が設定されます。最新の挿入が、このようなカラムがないテーブルに対して行われた場合は 0 になります。

@@identity の値は、接続に固有です。文が複数のローを挿入した場合、@@identity には最後に挿入されたローの IDENTITY 値が反映されます。文が IDENTITY カラムがないテーブルに影響を及ぼすと、@@identity に 0 が設定されます。

INSERT または SELECT INTO 文の実行に失敗したり、@@identity の値が保持されるトランザクションのロールバックがあっても、@@identity の値には影響を及ぼしません。挿入された文のコミットが失敗しても、@@identity には IDENTITY カラムに最後に挿入された値が保持されます。

@@identity とトリガー

挿入の実行によって、参照整合性アクションが起きたり、トリガーが起動されたりすると、@@identity はスタックのように動作します。たとえば、テーブル T1 (IDENTITY カラムまたは AUTOINCREMENT カラムがあるテーブル) に対する挿入によってトリガーが起動し、テーブル T2 (IDENTITY カラムまたは AUTOINCREMENT カラムがあるテーブル) にローが 1 つ挿入された場合、挿入を実行したアプリケーションやプロシージャーには、T1 に挿入された値が返されます。トリガー内では、T2 への挿入前には T1 の値が、挿入後には T2 の値が @@identity に入ります。トリガーが両方の値にアクセスする場合、トリガーはそれらの値をローカル変数にコピーできます。

標準と互換性

- **SQL/2008**　グローバル変数はベンダー拡張です。

コメント

コメントは、SQL 文または文ブロックに説明テキストを付加するために使用します。データベースサーバーは、コメントを実行しません。

次のコメントインジケーターが SQL Anywhere でサポートされています。

- **-- (二重ハイフン)**　データベースサーバーは、この行の残りの文字を無視します。これは、SQL/2008 のコメントインジケーターです。Interactive SQL と Sybase Central の [ストアドプロシージャー] ウィンドウで [Ctrl + マイナス記号] キーを押すと、このコメントインジケーターを追加および削除できます。

- **// (二重スラッシュ)**　二重スラッシュは、二重ハイフンと同じ意味です。Interactive SQL や Sybase Central の **[ストアドプロシージャー]** ウィンドウで [Ctrl + スラッシュ] キーを押すと、このコメントインジケーターを追加したり削除したりできます。

- **/* ... */ (スラッシュ - アスタリスク)**　2 つのコメントマーカーの間にある文字は、すべて無視されます。2 つのコメントマーカーは、同じ行にあっても、別の行にあってもかまいません。このスタイルで示されたコメントは、ネストできます。このスタイルは、「C スタイルコメント」とも呼ばれます。

例

次に、二重ハイフンのコメントの使用例を示します。

```
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;
```

次に、C スタイルコメントの使用例を示します。

```
/* Lists the names and employee IDs of employees
   who work in the sales department. */
CREATE VIEW SalesEmployees AS
  SELECT EmployeeID, Surname, GivenName
  FROM Employees
  WHERE DepartmentID = 200;
```

標準と互換性

- **SQL/2008**　コメントでの二重ハイフンの使用は、ANSI SQL/2008 標準のコア機能です。C スタイルの囲む形式のコメント (/* ...*/) は、SQL/2008 標準の SQL 言語機能 T351 です。二重スラッシュのコメント (//) は、ベンダー拡張としてサポートされます。

参照

- 「Interactive SQL のキーボードショートカット」『SQL Anywhere サーバー データベース管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL データ型

文字データ型

文字データ型は、文字、数字、記号などの文字列を格納するために使用します。

SQL Anywhere には 2 クラスの文字データ型と、そのデータ型を使用して定義されたドメインがいくつかあります。

- **CHAR、VARCHAR、LONG VARCHAR**　シングルバイトまたはマルチバイトの文字セットに格納される文字データ。多くの場合、データベースに格納されるプライマリ言語に最も対応するデータ型が選択されます。
- **NCHAR、NVARCHAR、LONG NVARCHAR**　UTF-8 Unicode の エンコーディングに格納される文字データ。データベースに格納されているプライマリ言語に関係なく、これらのデータ型を使用してすべての Unicode コードポイントを格納できます。
- **TEXT、UNIQUEIDENTIFIERSTR、XML**　他の文字データ型に基づくドメイン。

格納

すべての文字データ値は同じ方法で格納されます。デフォルトでは、1 つの値に最大で 128 バイトを格納できます。値が 128 バイトを超える場合、4 バイトのプレフィクスがデータベースページに格納され、全体の値は他の 1 つまたは複数のデータベースページに格納されます。このデフォルトサイズを制御するには、CREATE TABLE 文の INLINE 句と PREFIX 句を使用します。

参照

- 「CREATE TABLE 文」651 ページ
- 「string_rtruncation オプション」『SQL Anywhere サーバー データベース管理』

CHAR データ型

CHAR データ型は、32767 バイトまでの文字データを格納します。

構文

CHAR [**(** *max-length* [**CHAR** | **CHARACTER**] **)**]

パラメーター

- **max-length**　文字列の最大長。バイト長のセマンティックを使用する場合 (長さの部分に CHAR または CHARACTER を指定しない場合)、長さはバイト単位になります。また、範囲は 1 ～ 32767 にします。指定しない場合の長さは 1 です。

文字長のセマンティックを使用する場合 (長さの部分に CHAR または CHARACTER を指定する場合)、長さは文字単位になります。また、*max-length* を指定します。*max-length* は最大 32767 文字です。

備考

マルチバイト文字は CHAR 型に格納できますが、文字長のセマンティックを使用していなければ、宣言される長さは文字数ではなくバイト数です。

CHAR は CHARACTER と指定することもできます。どの構文を使用する場合でも、データ型は CHAR と記述されます。

セマンティック上、CHAR は VARCHAR と同じですが、型は異なります。SQL Anywhere では、CHAR は可変長型です。他のリレーショナルデータベース管理システムでは、CHAR は固定長型であり、データには *max-length* バイトまでブランクが埋め込まれて格納されます。SQL Anywhere では格納される文字データにブランクを埋め込みません。

CHAR カラムがどのように記述されるかは、クライアントインターフェイス、使用される文字セット、および文字長セマンティックが使用されるかどうかによって異なります。たとえば、Embedded SQL では、記述される長さはクライアント文字セットの最大バイト数となります。記述される長さが 32767 バイトを超える場合、カラムは DT_LONGVARCHAR 型として記述されます。次の表は、Embedded SQL の例と、DESCRIBE を実行した場合に返される結果を示します。

記述される型	データベースの文字セット	クライアントの文字セット	DESCRIBE の結果
CHAR(10)	Windows -1252	Windows -1252	DT_FIXCHAR 長さ 10
CHAR(10)	UTF-8	UTF-8	DT_FIXCHAR 長さ 10
CHAR(10)	Windows -1252	UTF-8	DT_FIXCHAR 長さ 30
CHAR(20000)	Windows-31J	UTF-8	DT_LONGVARCHAR
CHAR(10 CHAR)	Windows -1252	Windows-1252	DT_FIXCHAR 長さ 10
CHAR(10 CHAR)	UTF-8	UTF-8	DT_FIXCHAR 長さ 40

ODBC では、odbc_distinguish_char_and_varchar オプションに応じて、CHAR は SQL_CHAR または SQL_VARCHAR として記述されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「odbc_distinguish_char_and_varchar オプション」『SQL Anywhere サーバー データベース管理』
- 「VARCHAR データ型」91 ページ
- 「LONG VARCHAR データ型」88 ページ
- 「NCHAR データ型」88 ページ

標準と互換性

- **SQL/2008**　SQL/2008 と互換性があります。SQL/2008 標準では、文字長セマンティックがデフォルトですが、SQL Anywhere ではバイト長セマンティックがデフォルトです。大文字と小文字を区別しない照合のサポートと SQL Anywhere のブランク埋め込みに対するサポートのために、SQL 標準との間に、一部不整合があります。

  SQL/2008 標準では、SQL 言語機能 T061 として、明示的な文字長またはバイト長のセマンティックがサポートされています。

LONG NVARCHAR データ型

LONG NVARCHAR データ型は、任意の長さの Unicode 文字データを格納します。

構文

LONG NVARCHAR

備考

最大サイズは 2 GB から 1 バイトを差し引いた値です (2^{31} - 1)。

文字は UTF-8 に格納されます。各文字には 1 ～ 4 バイトが必要です。LONG NVARCHAR に格納できる最大文字数は 5 億を超え、格納される文字の長さによっては 20 億を超えることもあります。

Embedded SQL クライアントで LONG NVARCHAR カラムに DESCRIBE が実行される場合、db_change_nchar_charset 関数が呼び出されたかどうかに応じて、返されるデータ型は DT_LONGVARCHAR または DT_LONGNVARCHAR になります。

ODBC では、LONG NVARCHAR 式は SQL_WLONGVARCHAR として記述されます。

参照

- 「db_change_nchar_charset 関数」『SQL Anywhere サーバー プログラミング』
- 「NCHAR データ型」88 ページ
- 「NVARCHAR データ型」89 ページ
- 「LONG VARCHAR データ型」88 ページ

標準と互換性

- **SQL/2008**　LONG NVARCHAR を使用して各国の文字列を宣言できますが、この機能はベンダー拡張です。

LONG VARCHAR データ型

LONG VARCHAR データ型は、任意の長さの文字データを格納します。

構文

LONG VARCHAR

備考

最大サイズは 2 GB から 1 バイトを差し引いた値です (2^{31} - 1)。

マルチバイト文字は LONG VARCHAR として格納できますが、長さは文字数ではなくバイト数です。

参照

- 「CHAR データ型」85 ページ
- 「VARCHAR データ型」91 ページ
- 「LONG NVARCHAR データ型」87 ページ

標準と互換性

- **SQL/2008**　ラージオブジェクトのサポートは、SQL/2008 標準の SQL 言語機能 T041 です。

NCHAR データ型

NCHAR データ型は、32767 文字までの Unicode 文字データを格納します。

構文

NCHAR [(*max-length*)]

パラメーター

- **max-length**　文字列の文字単位での最大長。長さの範囲は 1 ～ 32767 です。指定しない場合の長さは 1 です。

備考

文字は UTF-8 エンコーディングを使用して格納されます。格納に必要な最大バイト数は、*max-length* を 4 倍した値です。ただし、通常、実際に使用される格納バイト数はより少ない値です。

NCHAR は NATIONAL CHAR または NATIONAL CHARACTER と指定することもできます。どの構文を使用する場合でも、データ型は NCHAR と記述されます。

Embedded SQL クライアントで NCHAR カラムに DESCRIBE が実行される場合、db_change_nchar_charset 関数が呼び出されたかどうかに応じて、返されるデータ型は DT_FIXCHAR または DT_NFIXCHAR になります。

また、Embedded SQL クライアントで NCHAR カラムに DESCRIBE が実行される場合、返される長さは、クライアントの NCHAR 文字セットの最大バイト長です。たとえば、西ヨーロッパ言語

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

の文字セット cp1252 を NCHAR 文字セットに使用する Embedded SQL クライアントの場合、NCHAR(10) カラムは、長さ 10 (10 文字の文字ごとに最大で 1 バイトを乗じた値) の DT_NFIXCHAR 型として記述されます。日本語の文字セット cp932 を使用する Embedded SQL クライアントの場合、同じ NCHAR(10) カラムは長さ 20 (10 文字の文字ごとに最大で 2 バイトを乗じた値) の DT_NFIXCHAR 型として記述されます。記述された長さが 32767 バイトを超える場合、カラムは DT_LONGNVARCHAR 型として記述されます。

セマンティック上、NCHAR は NVARCHAR と同じですが、型は異なります。SQL Anywhere では、NCHAR は可変長型です。他のリレーショナルデータベース管理システムでは、NCHAR は固定長型であり、データには *max-length* 文字までブランクが埋め込まれて格納されます。SQL Anywhere では格納される文字データにブランクを埋め込みません。

ODBC の場合、NCHAR は SQL_WCHAR として記述されます。

参照

- 「db_change_nchar_charset 関数」『SQL Anywhere サーバー プログラミング』
- 「CHAR データ型」85 ページ
- 「NVARCHAR データ型」89 ページ
- 「LONG NVARCHAR データ型」87 ページ

標準と互換性

- **SQL/2008**　各国の文字のサポートは、SQL/2008 標準の機能 F421 です。

NTEXT データ型

NTEXT データ型は、任意の長さの Unicode 文字データを格納します。

構文

NTEXT

備考

NTEXT はドメインです。LONG NVARCHAR として実装されます。

参照

- 「LONG NVARCHAR データ型」87 ページ
- 「TEXT データ型」90 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

NVARCHAR データ型

NVARCHAR データ型は、32767 文字までの Unicode 文字データを格納します。

構文

NVARCHAR [(*max-length*)]

パラメーター

- **max-length**　文字列の文字単位での最大長。長さの範囲は 1 ～ 32767 です。指定しない場合の長さは 1 です。

備考

文字は UTF-8 エンコーディングを使用して格納されます。格納に必要な最大バイト数は *max-length* を 4 倍した値ですが、通常、実際に使用される格納バイト数はより少ない値です。

NVARCHAR は、CHAR VARYING、NATIONAL CHAR VARYING、または NATIONAL CHARACTER VARYING と指定することもできます。どの構文を使用する場合でも、データ型は NVARCHAR と記述されます。

Embedded SQL クライアントで NVARCHAR カラムに DESCRIBE が実行される場合、db_change_nchar_charset 関数が呼び出されたかどうかに応じて、返されるデータ型は DT_VARCHAR または DT_NVARCHAR になります。

また、Embedded SQL クライアントで NVARCHAR カラムに DESCRIBE が実行される場合、返される長さは、クライアントの NCHAR 文字セットの最大バイト長です。たとえば、西ヨーロッパ言語の文字セット cp1252 を NCHAR 文字セットに使用する Embedded SQL クライアントの場合、NVARCHAR(10) カラムは、長さ 10 (10 文字の文字ごとに最大で 1 バイトを乗じた値) の DT_NVARCHAR 型として記述されます。日本語の文字セット cp932 を使用する Embedded SQL クライアントの場合、同じ NCHAR(10) カラムは長さ 20 (10 文字の文字ごとに最大で 2 バイトを乗じた値) の DT_NVARCHAR 型として記述されます。記述された長さが 32767 バイトを超える場合、カラムは DT_LONGNVARCHAR 型として記述されます。

ODBC の場合、NVARCHAR は SQL_WVARCHAR として記述されます。

参照

- 「db_change_nchar_charset 関数」『SQL Anywhere サーバー プログラミング』
- 「NCHAR データ型」88 ページ
- 「LONG NVARCHAR データ型」87 ページ
- 「VARCHAR データ型」91 ページ

標準と互換性

- **SQL/2008**　各国の文字のサポートは、SQL/2008 標準の SQL 言語機能 F421 です。

TEXT データ型

TEXT データ型は、任意の長さの文字データを格納します。

構文

TEXT

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

TEXT はドメインです。LONG VARCHAR として実装されます。

参照

- 「LONG VARCHAR データ型」88 ページ
- 「NTEXT データ型」89 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

UNIQUEIDENTIFIERSTR データ型

UNIQUEIDENTIFIERSTR は、CHAR(36) として実装されたドメインです。

構文

UNIQUEIDENTIFIERSTR

備考

Microsoft SQL Server の uniqueidentifier カラムをマッピングするとき、リモートデータアクセスに使用されます。

参照

- 「データ型変換：Microsoft SQL Server」『SQL Anywhere サーバー SQL の使用法』
- 「STRTOUUID 関数 [文字列]」359 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

VARCHAR データ型

VARCHAR データ型は、32767 バイトまでの文字データを格納します。

構文

VARCHAR [**(** *max-length* [**CHAR** | **CHARACTER**] **)**]

パラメーター

- **max-length**　文字列の最大長。バイト長のセマンティックを使用する場合 (長さの部分に CHAR または CHARACTER を指定**しない**場合)、長さはバイト単位になります。また、範囲は 1 ～ 32767 にします。指定しない場合の長さは 1 です。

  文字長のセマンティックを使用する場合 (長さの部分に CHAR または CHARACTER を指定する場合)、長さは文字単位になります。また、*max-length* を指定します。*max-length* は最大 32767 文字です。

備考

マルチバイト文字は VARCHAR として格納できますが、宣言される長さは文字数ではなくバイト数です。

VARCHAR は CHAR VARYING または CHARACTER VARYING と指定することもできます。どの構文を使用する場合でも、データ型は VARCHAR と記述されます。

VARCHAR カラムがどのように記述されるかは、クライアントインターフェイス、使用される文字セット、および文字長セマンティックが使用されるかどうかによって異なります。たとえば、Embedded SQL では、記述される長さはクライアント文字セットの最大バイト数となります。記述される長さが 32767 バイトを超える場合、カラムは DT_LONGVARCHAR 型として記述されます。次の表は、Embedded SQL の例と、DESCRIBE を実行した場合に返される結果を示します。

記述される型	データベースの文字セット	クライアントの文字セット	DESCRIBE の結果
VARCHAR(10)	Windows-1252	Windows-1252	DT_VARCHAR 長さ 10
VARCHAR(10)	UTF-8	UTF-8	DT_VARCHAR 長さ 10
VARCHAR(10)	Windows -1252	UTF-8	DT_VARCHAR 長さ 30
VARCHAR(20000)	Windows-31J	UTF-8	DT_LONGVARCHAR
VARCHAR(10 CHAR)	Windows-1252	Windows-1252	DT_VARCHAR 長さ 10
VARCHAR(10 CHAR)	UTF-8	UTF-8	DT_VARCHAR 長さ 40

ODBC の場合、VARCHAR は SQL_VARCHAR として記述されます。

参照

- 「CHAR データ型」85 ページ
- 「LONG VARCHAR データ型」88 ページ
- 「NVARCHAR データ型」89 ページ

標準と互換性

- **SQL/2008**　SQL/2008 と互換性があります。SQL/2008 標準では、文字長セマンティックがデフォルトですが、SQL Anywhere ではバイト長セマンティックがデフォルトです。大文字と小文字を区別しない照合のサポートと SQL Anywhere のブランク埋め込みに対するサポートのために、SQL 標準との間に、一部不整合があります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL/2008 標準では、SQL 言語機能 T061 として、明示的な文字長またはバイト長のセマンティックがサポートされています。

XML データ型

XML データ型は、任意の長さの文字データを格納します。XML ドキュメントを格納するために使用します。

構文

XML

備考

最大サイズは 2 GB から 1 バイトを差し引いた値です (2^{31} - 1)。

リレーショナルデータから要素内容を生成する場合、XML データ型は引用符で囲まれません。

文字列へのキャストまたは文字列からのキャストが可能な他のデータ型と XML データ型の間で、キャストができます。ただし、文字列を XML にキャストした場合、的確であるかどうかの検証は行われないことに注意してください。

Embedded SQL クライアントアプリケーションで、XML カラムに DESCRIBE が実行された場合、LONG VARCHAR と記述されます。

参照

- 「データベースにおける XML の使用」『SQL Anywhere サーバー SQL の使用法』
- 「リレーショナルデータベースにおける XML 文書の格納」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** XML データ型は、SQL/2008 標準の SQL 言語機能 X010 です。

数値データ型

数値データ型は、数値データを格納するために使用します。

NUMERIC データ型、DECIMAL データ型、さまざまな種類の INTEGER データ型は「真数値」データ型と呼ぶこともあります。これと対照的なのが、「概数値」データ型である FLOAT、DOUBLE、REAL です。

真数値データ型は、精度と小数点以下の桁数の値を指定できる型です。一方、概数値データ型は事前に定義された方法で格納されます。**真数値データだけが、算術演算後に指定した最少有効桁数に対して正確性が保証されます。**

1 より小さいデータ型の長さと精度は使用できません。

互換性

Transact-SQL identity カラムでは、scale = 0 の NUMERIC データ型のみ使用できます。

NUMERIC と DECIMAL のデータ型にデフォルトの精度と位取りの設定を使用する場合は注意が必要です。他のデータベースソリューションでは設定方法が異なることがあります。SQL Anywhere では、デフォルトの精度は 30、デフォルトの位取りは 6 です。

NUMERIC データ型と DECIMAL データ型では、デフォルトの精度と位取り設定を使用しないでください。これは、SQL Anywhere と Adaptive Server Enterprise で設定内容が異なるためです。SQL Anywhere では、デフォルトの精度は 30、デフォルトの位取りは 6 です。Adaptive Server Enterprise では、デフォルトの精度は 18、デフォルトの位取りは 0 です。

FLOAT (*p*) データ型は、*p* の値によって、REAL か DOUBLE のどちらかの同義語です。SQL Anywhere では、カットオフ値はプラットフォームによって異なりますが、どのプラットフォームでもカットオフ値は 16 以上です。

データベースオプションの設定によるデフォルトの変更については、「precision オプション」『SQL Anywhere サーバー データベース管理』と「scale オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

BIGINT データ型

BIGINT データ型は、8 バイトの記憶領域を必要とする整数である BIGINT を格納するために使用します。

構文

[UNSIGNED] BIGINT

備考

BIGINT データ型は真数値データ型です。精度は算術演算の後で保存されます。

BIGINT 値には 8 バイトの記憶領域が必要です。

符号付き BIGINT 値の範囲は、-2^{63} ～ $2^{63} - 1$、または -9223372036854775808 ～ 9223372036854775807 です。

符号なし BIGINT 値の範囲は、0 ～ $2^{64} - 1$、または 0 ～ 18446744073709551615 です。

デフォルトでは、このデータ型は符号付きです。

文字列を BIGINT に変換すると、前後のスペースは削除されます。先行文字が '+' の場合は無視されます。先行文字が '-' の場合は、残りの数字は負の数として解釈されます。先行の '0' 文字はスキップされ、残りの文字は整数値に変換されます。変換しようとしたデータ型の有効な範囲外の値である場合、文字列に不正な文字が含まれる場合、または文字列を整数値として復号化できない場合にエラーが返されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「BIT データ型」95 ページ
- 「INTEGER データ型」99 ページ
- 「SMALLINT データ型」101 ページ
- 「TINYINT データ型」102 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008** BIGINT データ型は、SQL/2008 標準の SQL 言語機能 T071 です。

BIT データ型

BIT データ型は、ビット (0 または 1) を格納するために使用します。

構文

BIT

備考

BIT は、0 または 1 の値を格納できる整数型です。

デフォルトでは、BIT データ型には NULL を入力できません。

BIT 値には 1 バイトの記憶領域が必要です。

文字列を BIT に変換すると、前後のスペースは削除されます。先行文字が '+' の場合は無視されます。先行文字が '-' の場合は、残りの数字は負の数として解釈されます。先行の '0' 文字はスキップされ、残りの文字は整数値に変換されます。変換しようとしたデータ型の有効な範囲外の値である場合、文字列に不正な文字が含まれる場合、または文字列を整数値として復号化できない場合にエラーが返されます。

参照

- 「BIGINT データ型」94 ページ
- 「INTEGER データ型」99 ページ
- 「SMALLINT データ型」101 ページ
- 「TINYINT データ型」102 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/1999** BIT データ型は、SQL/1999 標準の SQL 言語機能 F511 です。

- **SQL/2008** BIT データ型と BIT VARYING データ型は、SQL/2003 標準から削除されました。このため、SQL/2008 標準では、BIT データ型はベンダー拡張となっています。

DECIMAL データ型

DECIMAL データ型は、総桁数の *precision* と小数点以下の桁数の *scale* を持つ 10 進数です。

構文

DECIMAL [**(** *precision* [, *scale*] **)**]

パラメーター

- **precision**　式の桁数を指定する整数式。1 ～ 127 の範囲。デフォルト設定値は 30 です。
- **scale**　小数点以下の桁数を指定する整数式。0 ～ 127 の範囲。scale 値は precision 値以下にする必要があります。デフォルト設定値は 6 です。

  デフォルトは、データベースオプションを指定することによって変更できます。

備考

DECIMAL データ型は真数値データ型です。精度は、算術演算の後、最小の有効桁数まで保存されます。

10 進数に必要な記憶領域は次のように計算できます。

```
2 + int( (before + 1)/2 ) + int( (after + 1)/2 )
```

関数 int は引数の整数部分であり、before と after は小数点の前後の有効桁数です。記憶領域の計算の基準は、カラムの中で使われている最大精度と小数点以下の桁数ではなく、格納されている値を使います。

DECIMAL は DEC と指定することもできます。どの構文を使用する場合でも、データ型は DECIMAL と記述されます。

使用する精度が 20 以下で小数点以下の桁数が 0 の場合、代わりに整数データ型のいずれか (BIGINT、INTEGER、SMALLINT、または TINYINT) を使用できることがあります。整数値では、有効桁数が同じくらいの NUMERIC 値や DECIMAL 値よりも少ない記憶領域が使用されます。整数値の操作 (フェッチ、挿入など) や算術演算子は、通常、NUMERIC 値や DECIMAL 値の操作よりも速く実行されます。

セマンティック上、DECIMAL は NUMERIC と同じです。

> **注意**
> DECIMAL データ型のカラムまたは変数を作成するとき、これらのカラムまたは変数の精度または位取りがデータベースの精度および位取りの設定を超過する場合は 、値はデータベースの設定に合わせてトランケートされます。このため、DECIMAL として定義されているカラムまたは変数の値がトランケートされる場合は、精度と位取りがデータベースのオプション設定を超過していないかどうかを確認します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「FLOAT データ型」98 ページ
- 「REAL データ型」101 ページ
- 「DOUBLE データ型」97 ページ
- 「NUMERIC データ型」99 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ
- 「precision オプション」『SQL Anywhere サーバー データベース管理』
- 「scale オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** DECIMAL データ型と NUMERIC データ型は、SQL/2008 標準のコア機能です。

DOUBLE データ型

DOUBLE データ型は、倍精度の浮動小数点数を格納するために使用します。

構文

DOUBLE [PRECISION]

備考

DOUBLE データ型は、倍精度の浮動小数点数を保持します。これは概数値データ型です。算術演算後の丸め誤差がでます。概数値という DOUBLE 値の性質により、通常、DOUBLE 値を比較するときは等号を使用するクエリを避けてください。

DOUBLE 値には 8 バイトの記憶領域が必要です。

値の範囲は -1.79769313486231e+308 ～ 1.79769313486231e+308 であり、0 に最も近い最小の数値は 2.22507385850721e-308 です。DOUBLE として保持される値は、厳密には 15 有効桁数ですが、16 桁以上になると丸め誤差が出ます。

参照

- 「FLOAT データ型」98 ページ
- 「REAL データ型」101 ページ
- 「DECIMAL データ型」96 ページ
- 「NUMERIC データ型」99 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ
- 「数値セットの変換」135 ページ

標準と互換性

- **SQL/2008** DOUBLE PRECISION 型は、SQL/2008 標準のコア機能です。

FLOAT データ型

FLOAT データ型は、単精度または倍精度の浮動小数点数を格納するために使用します。

構文

FLOAT [(*precision*)]

パラメーター

- **precision**　仮数部の桁数を指定する整数式。仮数部は、常用対数の小数部です。たとえば、対数 5.63428 の仮数は 0.63428 です。IEEE 標準 754 による浮動小数点精度は、次のとおりです。

提供されている精度値	10 進数精度	等価の SQL データ型	記憶サイズ
1-24	7 桁	REAL	4 バイト
25-53	15 桁	DOUBLE	8 バイト

備考

FLOAT (*precision*) データ型を使ってカラムを作成すると、すべてのプラットフォームのカラムには、少なくとも指定した最小精度で値が格納されることが保証されます。一方、REAL と DOUBLE の場合、プラットフォームに依存しない最小精度は保証されません。

precision を指定しない場合、FLOAT データ型は単精度の浮動小数点数です。これは REAL データ型と等価で、4 バイトの記憶領域を必要とします。

precision を指定する場合、FLOAT データ型は指定した精度の値によって、単精度か倍精度のいずれかになります。REAL になるか DOUBLE になるかは、プラットフォームによって異なります。単精度の FLOAT 値は、4 バイトの記憶領域を必要とし、倍精度の FLOAT 値は、8 バイトを必要とします。

FLOAT データ型は概数値データ型です。算術演算後の丸め誤差がでます。概数値という FLOAT 値の性質により、通常、FLOAT 値を比較するときは等号を使用するクエリを避けてください。

参照

- 「DOUBLE データ型」97 ページ
- 「REAL データ型」101 ページ
- 「DECIMAL データ型」96 ページ
- 「NUMERIC データ型」99 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008**　FLOAT 型は、SQL/2008 標準のコア機能です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

INTEGER データ型

INTEGER データ型は、4 バイトの記憶領域を必要とする整数を格納するために使用します。

構文

[UNSIGNED] INTEGER

備考

INTEGER データ型は真数値データ型で、精度は算術演算の後で保存されます。

UNSIGNED を指定した場合、整数に負の数を割り当てることはできません。デフォルトでは、このデータ型は符号付きです。

符号付き整数値の範囲は、-2^{31} ～ $2^{31} - 1$、または -2147483648 ～ 2147483647 です。

符号なし整数の範囲は、0 ～ $2^{32} - 1$、または 0 ～ 4294967295 です。

INTEGER は INT と指定することもできます。どの構文を使用する場合でも、データ型は INTEGER と記述されます。

文字列を INTEGER に変換すると、前後のスペースは削除されます。先行文字が '+' の場合は無視されます。先行文字が '-' の場合は、残りの数字は負の数として解釈されます。先行の '0' 文字はスキップされ、残りの文字は整数値に変換されます。変換しようとしたデータ型の有効な範囲外の値である場合、文字列に不正な文字が含まれる場合、または文字列を整数値として復号化できない場合にエラーが返されます。

参照

- 「BIGINT データ型」94 ページ
- 「BIT データ型」95 ページ
- 「SMALLINT データ型」101 ページ
- 「TINYINT データ型」102 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008**　INTEGER 型は、SQL/2008 標準のコア機能です。UNSIGNED キーワードはベンダー拡張です。

NUMERIC データ型

NUMERIC データ型は、総桁数の *precision* と小数点以下の桁数の *scale* を持つ 10 進数を格納するために使用します。

構文

NUMERIC [(*precision* [, *scale*] **)]**

パラメーター

- **precision**　式の桁数を指定する整数式。1 ～ 127 の範囲。デフォルト設定値は 30 です。
- **scale**　小数点以下の桁数を指定する整数式。0 ～ 127 の範囲。scale 値は precision 値以下にする必要があります。デフォルト設定値は 6 です。

備考

NUMERIC データ型は真数値データ型です。精度は、算術演算の後、最小の有効桁数まで保存されます。

10 進数を格納するために必要なバイト数は、次のように計算できます。

```
2 + INT( (BEFORE+1)/2 ) + INT( (AFTER+1)/2 )
```

INT 関数は引数の整数部分であり、BEFORE と AFTER は小数点の前後の有効桁数です。記憶領域の計算の基準は、カラムの中で使われている最大精度と小数点以下の桁数ではなく、格納されている値を使います。

使用する精度が 20 以下で小数点以下の桁数が 0 の場合、代わりに整数データ型のいずれか (BIGINT、INTEGER、SMALLINT、または TINYINT) を使用できることがあります。整数値では、有効桁数が同じくらいの NUMERIC 値や DECIMAL 値よりも少ない記憶領域が使用されます。整数値の操作 (フェッチ、挿入など) や算術演算子は、通常、NUMERIC 値や DECIMAL 値の操作よりも速く実行されます。

セマンティック上、NUMERIC は DECIMAL と同じです。

注意
NUMERIC データ型のカラムまたは変数を作成するとき、これらのカラムまたは変数の精度または位取りがデータベースの精度および位取りの設定を超過する場合は 、値はデータベースの設定に合わせてトランケートされます。このため、NUMERIC として定義されているカラムまたは変数の値がトランケートされる場合は、精度と位取りがデータベースのオプション設定を超過していないかどうかを確認します。

参照

- 「FLOAT データ型」98 ページ
- 「REAL データ型」101 ページ
- 「DOUBLE データ型」97 ページ
- 「DECIMAL データ型」96 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ
- 「数値セットの変換」135 ページ
- 「precision オプション」『SQL Anywhere サーバー データベース管理』
- 「scale オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　scale オプションが 0 に設定されている場合、SQL/2008 と互換性があります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

REAL データ型

REAL データ型は、4 バイトで格納される単精度浮動小数点数を格納するために使用します。

構文

REAL

備考

REAL データ型は、単精度の浮動小数点数を保持します。これは概数値データ型です。算術演算後の丸め誤差がでます。概数値という REAL 値の性質により、通常、REAL 値を比較するときは等号を使用するクエリを避けてください。

REAL 値には 4 バイトの記憶領域が必要です。

値の範囲は、 -3.402823e+38 ～ 3.402823e+38 であり、0 に最も近い最小の数値は 1.175494351e-38 です。REAL として保持される値は、厳密には 7 有効桁数ですが、7 桁以上になると丸め誤差が出ます。

参照

- 「DOUBLE データ型」97 ページ
- 「FLOAT データ型」98 ページ
- 「DECIMAL データ型」96 ページ
- 「NUMERIC データ型」99 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008**　REAL データ型は、SQL/2008 標準のコア機能です。

SMALLINT データ型

SMALLINT データ型は、2 バイトの記憶領域を必要とする整数を格納するために使用します。

構文

[UNSIGNED] SMALLINT

備考

SMALLINT データ型は真数値データ型です。精度は算術演算の後で保存されます。2 バイトの記憶領域を必要とします。

符号付き SMALLINT 値の範囲は、-2^{15} ～ 2^{15} - 1、または -32768 ～ 32767 です。

符号なし SMALLINT 値の範囲は、0 ～ 2^{16} - 1、または 0 ～ 65535 です。

文字列を SMALLINT に変換すると、前後のスペースは削除されます。先行文字が '+' の場合は無視されます。先行文字が '-' の場合は、残りの数字は負の数として解釈されます。先行の '0' 文字

はスキップされ、残りの文字は整数値に変換されます。変換しようとしたデータ型の有効な範囲外の値である場合、文字列に不正な文字が含まれる場合、または文字列を整数値として復号化できない場合にエラーが返されます。

参照

- 「BIGINT データ型」94 ページ
- 「BIT データ型」95 ページ
- 「INTEGER データ型」99 ページ
- 「TINYINT データ型」102 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008**　SQL/2008 と互換性があります。UNSIGNED キーワードはベンダー拡張です。

TINYINT データ型

TINYINT データ型は、1 バイトの記憶領域を必要とする符号なし整数を格納するために使用します。

構文

[UNSIGNED] TINYINT

備考

TINYINT データ型は真数値データ型です。精度は算術演算の後で保存されます。

明示的に TINYINT を UNSIGNED として指定できますが、この型は必ず符号なしであるため、UNSIGNED 変更子は効力を持ちません。

TINYINT 値の範囲は、0 ～ 2^8 - 1、または 0 ～ 255 です。

Embedded SQL では、TINYINT カラムを char または unsigned char と定義された変数にフェッチしないでください。これは、カラムの値を文字列に変換し、最初のバイトをプログラムの変数に割り当てようとすることになるからです。TINYINT カラムは 2 バイトまたは 4 バイト整数にフェッチしてください。また、TINYINT 値を C で書かれたアプリケーションからデータベースに送るには、C 変数の型に整数を指定してください。

文字列を TINYINT に変換すると、前後のスペースは削除されます。先行文字が '+' の場合は無視されます。先行文字が '-' の場合は、残りの数字は負の数として解釈されます。先行の '0' 文字はスキップされ、残りの文字は整数値に変換されます。変換しようとしたデータ型の有効な範囲外の値である場合、文字列に不正な文字が含まれる場合、または文字列を整数値として復号化できない場合にエラーが返されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「BIGINT データ型」94 ページ
- 「BIT データ型」95 ページ
- 「INTEGER データ型」99 ページ
- 「SMALLINT データ型」101 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

通貨データ型

通貨データ型は、通貨データを格納するために使用します。

MONEY データ型

MONEY データ型は、通貨データを格納します。

構文

MONEY

備考

MONEY は NUMERIC(19,4) として実装されるドメインです。

参照

- 「SMALLMONEY データ型」103 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

SMALLMONEY データ型

SMALLMONEY データ型は、100 万通貨単位未満の通貨データを格納するために使用します。

構文

SMALLMONEY

備考

SMALLMONEY は NUMERIC(10,4) として実装されるドメインです。

参照

- 「MONEY データ型」103 ページ
- 「数値関数」146 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

ビット配列データ型

ビット配列は文字列に似ていますが、使用される要素はビットデータであり、文字ではなく 0 (ゼロ) と 1 になります。通常、ビット配列は、ブール値の文字列を保持するのに使用されます。

SQL Anywhere がサポートするビット配列データ型には、VARBIT と LONG VARBIT があります。

LONG VARBIT データ型

LONG VARBIT データ型は、任意の長さのビット配列を格納するために使用します。

構文

LONG VARBIT

備考

任意の長さのビット (1 と 0) 配列または 32767 ビットを超えるビット配列を格納するときに使用します。

LONG VARBIT は LONG BIT VARYING と指定することもできます。どの構文を使用する場合でも、データ型は LONG VARBIT と記述されます。

参照

- 「BIT データ型」95 ページ
- 「VARBIT データ型」105 ページ
- 「ビット配列の変換」134 ページ
- 「ビット配列関数」141 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

VARBIT データ型

VARBIT データ型は、長さが 32767 未満のビット配列を格納するときに使用します。

構文

VARBIT [(*max-length*)]

パラメーター

- **max-length**　ビット配列のバイト単位での最大長。長さの範囲は 1 ～ 32767 です。指定しない場合の長さは 1 です。

備考

VARBIT は BIT VARYING と指定することもできます。どの構文を使用する場合でも、データ型は VARBIT と記述されます。

参照

- 「BIT データ型」95 ページ
- 「LONG VARBIT データ型」104 ページ
- 「ビット配列の変換」134 ページ
- 「ビット配列関数」141 ページ
- 「集合関数」139 ページ
- 「ビット処理演算子」11 ページ

標準と互換性

- **SQL/1999**　BIT VARYING データ型は、SQL/1999 標準の SQL 言語機能 F511 です。
- **SQL/2008**　BIT データ型と BIT VARYING データ型は、SQL/2003 標準から削除されました。このため、SQL/2008 標準では、BIT VARYING データ型はベンダー拡張です。

日付と時刻データ型

次のリストに、日付を処理する方法の概要を示します。

- SQL Anywhere は、算出した値が別の世紀にかかるかどうかにかかわらず、日付に関する有効な算術や論理演算に常に正しい値を返します。
- SQL Anywhere の日付の内部記憶領域には、年の値の世紀にあたる部分が常に明示的に含まれています。
- SQL Anywhere の操作は、現在の日付を含むどのような戻り値にも左右されません。
- 日付の値は、いつでも 4 桁の西暦で出力できます。

日付と時刻が格納されるしくみ

日付と時刻は、次のいずれかのデータ型を使用して SQL Anywhere データベースに格納されます。

データ型	内容	記憶領域	指定できる値の範囲
DATE	暦日 (年、月、日)	4 バイト	0001-01-01 ～ 9999-12-31 の日付。
TIME	時刻 (時、分、秒、秒の小数部分 (小数点第 6 位まで))	8 バイト	00:00:00.000000 ～ 24:00:00.000000 の時刻。
TIMESTAMP	日の暦日と時刻 (年、月、日、時、分、秒、秒の小数部分 (小数点第 6 位まで))	8 バイト	0001-01-01 ～ 9999-12-31 の日付 (TIMESTAMP の時と分の精度については、1600-02-28 23:59:59 より前と 7911-01-01 00:00:00 より後の部分は削除されます)。
TIMESTAMP WITH TIME ZONE	暦日、時刻、タイムゾーンオフセット (年、月、日、時、分、秒、秒の小数部分 (小数点第 6 位まで)、時間単位と分単位のタイムゾーンオフセット)	10 バイト	0001-01-01 ～ 9999-12-31 の日付 (TIMESTAMP WITH TIME ZONE の時と分の精度については、1600-02-28 23:59:59 より前と 7911-01-01 00:00:00 より後の部分は削除されます)。-14:59 ～ +14:59 のゾーンオフセット。

日付と時刻をデータベースに送信する

日付と時刻は次のいずれかの方法でデータベースに送信できます。

- インターフェイスを使うときは、文字列として
- ODBC または OLE DB を使うときは、バイナリ値として (例 : ODBC TIMESTAMP_STRUCT 構造体を使用する)
- Embedded SQL を使うときは、SQLDATETIME 構造体として

タイムゾーンオフセット付きの日付と時刻は、データベースに文字列としてのみ送信できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

日付フォーマット

日付を文字列 (DATE データ型用) として、または文字列の一部 (TIMESTAMP データ型または TIMESTAMP WITH TIME ZONE データ型用) としてデータベースに送信する場合、文字列は、日付と時刻の表現の国際標準である ISO 8601 によって記述されるものなど、いくつものさまざまな方法で指定できます。

日付は、次の ISO 8601 フォーマットの 1 つで指定できます。

- **暦日**　暦日フォーマットは YYYY-MM-DD であり、YYYY はグレゴリオ暦の年、MM は 01 (1 月) から 12 (12 月) までの月、DD は 01 から 31 までの日になります。たとえば、'2010-04-01' は、2010 年 4 月 1 日を表します。ISO 8601 は区切り文字を要求しません。そのため、'20100401' も 2010 年 4 月 1 日を表します。

ISO 暦日	フォーマット	例
基本	YYYYMMDD	20100401
拡張	YYYY-MM-DD	2010-04-01

- **週日**　もう 1 つの ISO 日付フォーマットが週日です。このフォーマットは YYYY-Www-D であり、YYYY はグレゴリオ暦の年、W は文字 **W**、ww は 01 (第 1 週) から 52 または 53 (最終週) までの年内の暦週、D は 1 (月曜日) から 7 (日曜日) までの週内の曜日です。たとえば、'2010-W13-4' は、2010 年の 13 番目の週の 4 番目の日 (2010 年 4 月 1 日) を表します。ISO 8601 は区切り文字を要求しません。そのため、'2010W134' も 2010 年の 13 番目の週の 4 番目の日を表します。精度を下げた場合、1 桁目を表現から省略できます ('2010W13' は 2010 年 3 月 29 日を表します)。

ISO 週日	フォーマット	例
基本	YYYYWwwD	2010W134
拡張	YYYY-Www-D	2010-W13-4

- **年内の日の番号**　最後の ISO 日付フォーマットは、年内の日の番号です。フォーマットは YYYY-DDD であり、YYYY はグレゴリオ暦の年、DDD はその暦年内で第何日目かを指します。たとえば、'2010-091' は 2010 年 4 月 1 日を表します。ISO 8601 は区切り文字を要求しません。たとえば、'2010091'も 2010 年 4 月 1 日を表します。年内の日の番号の最大値は、閏年用の 366 です。たとえば、'2008366' は、2008 年の最後の日 (2008 年 12 月 31 日) を表します。

ISO 年内の日の番号	フォーマット	例
基本	YYYYDDD	2010091
拡張	YYYY-DDD	2010-091

これら以外の日付フォーマットもサポートされています。SQL Anywhere は、日付を含む文字列を非常に柔軟に解釈します。あいまいである場合は、date_order および nearest_century データベースオプション設定に従って日付値の解釈が行われます。たとえば、date_order 設定に従い、

'02/05/2002' は、データベースサーバーの解釈によって、5 月 2 日 (DMY) とすることも、2 月 5 日 (MDY) とすることも、無効な値 (YMD) とすることもできます。

nearest_century 設定は、2 桁の年の値を 20 世紀と 21 世紀のどちらとして解釈するかを決定します。たとえば、文字列 '02/05/10' において、date_order 設定が 02 と 10 のどちらを年として解釈するのかを決定し、nearest_century 設定が 02 で 1902 年と 2002 年のどちらか、または 10 で 1910 年 2010 年のどちらを表すのかを決定します。nearest_century オプションの値は、2 桁の年数の解釈に影響します。nearest_century より少ない値には 2000 が加算され、その他のすべての値には 1900 が加算されます。このオプションのデフォルト値は 50 です。そのため、デフォルトで 50 年は 1950 年、49 年は 2049 年と解釈されます。

次の表は、指定された date_order 設定と 50 の nearest_century 設定を使って、2010 年の 4 月 1 日をどのように指定できるのかを示しています。

date_order	フォーマット	例
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

ISO 8601 フォーマットはあいまいではなく、date_order と nearest_century のユーザー設定の影響を受けないので、ISO 8601 を使用することをおすすめします。

また、日付は月の名前を使って指定することもできます。たとえば、'2010 April 01'、'April 1, 2010'、'1 April 2010' などがあります。年があいまいに指定された場合、date_order オプションを使って、年と日の部分を分解できます。そのため、'01 April 10' は、date_order が 'YMD' のときは 2001 年 4 月 10 日と解釈され、date_order が 'DMY' のときは 2010 年 4 月 1 日と解釈されます。

日付の中の年の値の範囲は 0001 ～ 9999 です。SQL Anywhere の最小日付は 0001-01-01 です。

文字列に部分的な日付指定だけがある場合、デフォルト値を使って日付が満たされます。次のデフォルトを使います。

- **年**　年が何も指定されない場合 ('April 1' など) は、現在の年が使用されます。
- **月**　年と月が何も指定されない（'23:59:59'など)の場合は、現在の月が使用されます。また、年が指定された場合（'2010'など）は、01 が使用されます。
- **日**　年と月が何も指定されない（'23:59:59'など)の場合は、現在の日が使用されます。また、月が指定された場合（'April'など）は、01 が使用されます。

次の例では、現在の日付から日付値が構築されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT CAST('23:59:59' AS TIMESTAMP)
```

時間フォーマット

時刻は、24 時間管理システムを使って ISO 8601 フォーマットで指定できます。これは hh:mm:ss であり、hh は真夜中から経過した完全な時間数、mm は時間の始めからの完全な分数、ss は分のはじめからの完全な秒数です。たとえば、'23:59:59' は真夜中より 1 秒前の時刻を表します。

ISO 8601 標準では、秒と分を省略できます。たとえば、'23:59' は真夜中より 60 秒前の時刻を表します。

ISO 8601 標準は、また、秒単位に小数を含めることができます。小数の秒は、カンマ (,) またはピリオド (.) を使って指定されます。秒の小数は、最大小数第 6 位まで格納されます。たとえば、'23:59:59,500000' と '23:59:59.500000' は、どちらも真夜中より 1/2 秒前の時刻を表します。SQL Anywhere は分または時間の小数をサポートしません。

ISO 8601 は、時刻が日付仕様に含まれるときには、コロン区切り文字を必要としません。たとえば、'235959' は真夜中より 1 秒前の時刻を表します。

日の最大時刻は '24:00:00' です。これは真夜中を表します。日付と組み合わせると、これは真夜中、または次の日の 00:00:00 を表します。たとえば、'2010-04-01 24:00:00' は '2010-04-02 00:00:00' と同じです。

ISO 時刻	フォーマット	例
基本 (日付あり)	hhmmss.ssssss	20100401 235959.500000
基本 (日付あり)	hhmmss,ssssss	20100401 235959,500000
拡張	hh:mm:ss.ssssss	23:59:59.500000
拡張	hh:mm:ss,ssssss	23:59:59,500000

ISO 以外の AM と PM の指示子もサポートされています。たとえば、'11:59:59 PM' は '23:59:59' と同じです。

AM/PM	フォーマット	例
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

時刻フォーマットのある日付

ISO 8601 では、日付と時刻で、スペースまたは文字 **T** を使用して組み合わせることができます。たとえば、'2010-04-01 23:59:59' と '2010-04-01T23:59:59' はどちらも 2010 年 4 月 1 日の真夜中よ

りも 1 秒前を表します。ハイフンとコロンの区切り文字は省略できます。たとえば、'20100401T235959' も同じ日付と時刻を表します。

SQL Anywhere は、日付と時刻のフォーマットの基本と拡張の組み合わせをサポートします。たとえば、'20100401T23:59:59' は基本フォーマットと拡張フォーマットの両方を組み合わせています。

タイムゾーンフォーマット

ISO 8601 は、また、日文字列の日付と時刻にタイムゾーンオフセットを追加することを許可します。フォーマットは次のいずれかになります。

- **Z** (ズールー) 日付と時刻は、協定世界時 (UTC: Coordinated Universal Time) 単位になります。たとえば、'2010-04-01 23:00:00Z' は、2010 年 4 月 1 日の協定世界時 11:00 PM を表します。
- **+hh:mm** 指定された日付と時刻は、UTC より、指定された時間と分だけ進んでいます。たとえば、'2010-04-01 23:00:00+04:00' は、UTC から 4 時間東のタイムゾーンでの 2010 年 4 月 1 日の 11:00 PM を表します。
- **-hh:mm** 指定された日付と時刻は、UTC より、指定された時間と分だけ遅れています。たとえば、'2010-04-01 23:00:00-05:00' は、UTC から 5 時間西のタイムゾーンでの 2010 年 4 月 1 日の 11:00 PM を表します。

分が 0 の場合、タイムゾーンオフセットで指定する必要はありません。また、スペースをタイムゾーンオフセットの前に含めることもできます。たとえば、'2010-04-01 23:00:00 -03:30' は、UTC から 3 時間半西のタイムゾーンでの 2010 年 4 月 1 日の 11:00 PM を表します。

ISO タイムゾーン	フォーマット	例
基本	Z	20100401 235959Z
基本	+hhmm	20100401 235959+0400
基本	+hh	20100401 235959+04
基本	-hhmm	20100401 235959-0500
基本	-hh	20100401 235959-05
基本	T を小数に使用	20100401T235959.50-0330
拡張	Z	2010-04-01 23:59:59Z
拡張	+hh:mm	2010-04-01 23:59:59+04:00
拡張	-hh:mm	2010-04-01 23:59:59-05:00
拡張	T を小数に使用	2010-04-01T23:59:59.50-03:30

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL Anywhere は、日付、時刻、タイムゾーンのフォーマットの基本と拡張の組み合わせをサポートします。たとえば、'20100401T23:59:59-05' は基本フォーマットと拡張フォーマットの両方を組み合わせています。

参照

- 「日付と時刻の比較」130 ページ
- 「DATE データ型」112 ページ
- 「date_format オプション」『SQL Anywhere サーバー データベース管理』
- 「nearest_century オプション」『SQL Anywhere サーバー データベース管理』
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ

日付と時刻のデータベースからの取得

次のいずれかの方法で、日付と時刻をデータベースから検索できます。

- インターフェイスを使うときは、文字列として
- ODBC または OLE DB を使うときは、バイナリ値として (例 : ODBC TIMESTAMP_STRUCT 構造体を使用する)
- Embedded SQL を使うときは、SQLDATETIME 構造体として

タイムゾーンオフセット付きの日付と時刻は、データベースから文字列としてのみ取得できます。

(タイムゾーンオフセット付きかどうかにかかわらず) 日付または時刻を文字列として検索する場合は、データベースオプション date_format、time_format、timestamp_format、および timestamp_with_time_zone_format によって指定されているフォーマットで検索します。

日付については次の算術演算子を使います。

- **timestamp + integer**　日付またはタイムスタンプに指定日数を加算する。
- **timestamp - integer**　日付またはタイムスタンプから指定日数を減算する。
- **date - date**　2 つの日付またはタイムスタンプの間の日数を計算する。
- **date + time**　与えられた日付と時刻を結合するタイムスタンプを作成する。

閏年

SQL Anywhere では、広く認められているアルゴリズムを使用して閏年の決定を行います。このアルゴリズムを使用すると、4 で割り切れる年は閏年と見なされます。1900 年などの世紀にあたる年の場合は、400 で割り切れれば閏年となります。

SQL Anywhere ではすべての閏年を正確に処理します。たとえば、次の SQL 文は "火曜日" という値を返します。

```
SELECT DAYNAME('2000-02-29');
```

SQL Anywhere は閏年 2000 年 2 月 29 日を日付として認め、この日付を考慮して曜日を判断します。

ただし、次の文は SQL Anywhere では拒否されます。

```
SELECT DAYNAME('2001-02-29');
```

エラーメッセージ「値 '2001-02-29' を日付に変換できません」が表示されます。これは、2 月 29 日が 2001 年には存在しないためです。

参照

- 「SET OPTION 文」913 ページ
- 「日付と時刻関数」142 ページ

DATE データ型

DATE データ型は、年、月、日などの暦日を格納するために使用します。

構文

DATE

備考

アプリケーションが文字列として DATE 値を取得するときのフォーマットは、date_format オプションの設定によって決まります。たとえば、2010 年 7 月 19 日を表す DATE 値は、date_format オプション設定に従い、アプリケーションに 2010/07/19 または Jul 19, 2010 として返されます。

DATE 値には 4 バイトの記憶領域が必要です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「日付と時刻関数」142 ページ
- 「日付フォーマット」107 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「date_format オプション」『SQL Anywhere サーバー データベース管理』
- 「date_order オプション」『SQL Anywhere サーバー データベース管理』
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「nearest_century オプション」『SQL Anywhere サーバー データベース管理』
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　SQL/2008 と互換性があります。
- **Transact-SQL**　Adaptive Server Enterprise によってサポートされます。

DATETIME データ型

DATETIME は、TIMESTAMP として実装されるドメインで、日付と時刻の情報を格納するために使用します。DATETIME は Transact-SQL の型です。

構文

DATETIME

備考

アプリケーションが文字列として DATETIME 値を取得するときのフォーマットは、timestamp_format オプションの設定によって決まります。たとえば、DATETIME 値 2010/04/01T23:59:59.999999 は、timestamp_format オプション設定に従い、アプリケーションに 2010/04/01 23:59:59 または April 1, 2010 23:59:59.999999 として返されます。

DATETIME 値には 8 バイトの記憶領域が必要です。

DATETIME データ型では、DATE 型と同じく 0001 ～ 9999 年の日付を格納できますが、DATETIME データ型の有効範囲は 1600-02-28 23:59:59 ～ 7911-01-01 00:00:00 です。DATETIME 値では、この範囲より前または後の時間および分の部分は保持されません。

注意
DATETIME 値の精度が下がると、分または秒に関連する組み込み関数が無効な結果を生成します。

DATETIME 値を DATETIMEOFFSET に変換する場合、接続の time_zone_adjustment 設定が結果のタイムゾーンオフセットに使用されます。つまり、値はその接続に「ローカル」なものであると見なされます。DATETIMEOFFSET 値を DATETIME に変換すると、オフセットは廃棄されます。

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「日付と時刻関数」142 ページ
- 「日付と時刻をデータベースに送信する」106 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「date_format オプション」『SQL Anywhere サーバー データベース管理』
- 「date_order オプション」『SQL Anywhere サーバー データベース管理』
- 「DATE データ型」112 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「timestamp_format オプション」『SQL Anywhere サーバー データベース管理』
- 「nearest_century オプション」『SQL Anywhere サーバー データベース管理』
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　Adaptive Server Enterprise によって使用されるのは、TIMESTAMP ではなく DATETIME です。Adaptive Server Enterprise の DATETIME 型では、1753 年 1 月 1 日から 9999 年 12 月 31 日までの日付がサポートされ、値の時刻の部分でサポートされている精度はより低くなっています。SQL Anywhere では、DATETIME は、これらの制約なしに TIMESTAMP として実装されます。SQL Anywhere と Adaptive Server Enterprise の間でデータを移行する場合は、これらの違いに注意してください。

DATETIMEOFFSET データ型

DATETIMEOFFSET データ型は TIMESTAMP WITH TIME ZONE のエイリアスであり、日付、時刻、およびタイムゾーン情報の格納に使用されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

DATETIMEOFFSET

備考

DATETIMEOFFSET 値は、年、月、日、時、分、秒、秒の小数位、および協定世界時 (UTC: Coordinated Universal Time) との差異を示す分数を含んでいます。秒以下は 6 桁まで格納されます。

アプリケーションが文字列として DATETIMEOFFSET 値を取得するときのフォーマットは、timestamp_with_time_zone_format オプションの設定によって決まります。たとえば、DATETIMEOFFSET 値 2010/04/01T23:59:59.999999-6:00 は、timestamp_with_time_zone_format オプション設定に従い、アプリケーションに 2010/04/01 23:59:59 -06:00 または April 1, 2010 23:59:59.999999 -06:00 として返されます。

DATETIMEOFFSET 値には 10 バイトの記憶領域が必要です。

DATETIMEOFFSET データ型では、DATE 型と同じく 0001 ～ 9999 年の日付を格納できますが、DATETIMEOFFSET データ型の有効範囲は 1600-02-28 23:59:59 ～ 7911-01-01 00:00:00 です。DATETIMEOFFSET 値では、この範囲より前または後の時間および分の部分は保持されません。

DATETIMEOFFSET は計算カラムまたはマテリアライズドビューには使用しないでください。時刻を制御する time_zone_adjustment オプションの値が、場所に基づく接続と日付によって異なるためです。

2 つの DATETIMEOFFSET 値が UTC において同じ時刻を表す場合、適用されている TIME ZONE オフセットに関係なく同じ時刻と見なされます。たとえば、次の文では、結果が同じと見なされるため、**Yes** が返されます。

```
IF CAST('2009-07-15 08:00:00 -08:00' AS DATETIMEOFFSET) =
CAST('2009-07-15 11:00:00 -05:00' AS DATETIMEOFFSET) THEN
SELECT  'Yes'
ELSE
SELECT 'No'
END IF;
```

DATETIMEOFFSET 値からタイムゾーンオフセットを省略すると、タイムスタンプが日付と時刻を標準時間または夏時間のいずれで表していても、クライアントの現在の UTC オフセットがデフォルト設定されます。たとえば、クライアントが東部標準タイムゾーンに属しており、夏時間が有効なときに次の文を実行した場合、大西洋標準タイムゾーン (UTC から -4 時間) に該当するタイムゾーンでタイムスタンプが返されます。

```
SELECT CAST('2009/01/30 12:34:55' AS DATETIMEOFFSET);
```

DATETIMEOFFSET 値をタイムゾーン付きではないタイムスタンプと比較することはおすすめしません。これは、クライアントのデフォルトのタイムゾーンオフセットは、クライアントの地理的な場所および日付によって異なるためです。

クライアントの現在のタイムゾーンオフセットを分単位で判別するには、次の文を実行します。

```
SELECT CONNECTION_PROPERTY( 'TimeZoneAdjustment' );
```

> **注意**
> TimeZoneAdjustment 接続プロパティは、Ultra Light データベースではサポートされません。

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「日付と時刻関数」142 ページ
- 「日付と時刻をデータベースに送信する」106 ページ
- 「DATE データ型」112 ページ
- 「DATETIME データ型」113 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「timestamp_with_time_zone_format オプション」『SQL Anywhere サーバー データベース管理』
- 「nearest_century オプション」『SQL Anywhere サーバー データベース管理』
- 「date_order オプション」『SQL Anywhere サーバー データベース管理』
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　DATETIMEOFFSET の特定の使用方法はベンダー拡張です。SQL/2008 と互換性を保つには、TIMESTAMP WITH TIME ZONE を使用します。TIMESTAMP WITH TIME ZONE 型は、SQL/2008 標準のオプションの SQL 言語機能 F411 です。

SMALLDATETIME データ型

SMALLDATETIME は、TIMESTAMP として実装されるドメインで、日付と時刻の情報を格納するために使用します。SMALLDATETIME は Transact-SQL の型です。

構文

SMALLDATETIME

備考

日付と時刻の仕様の詳細については、「日付と時刻をデータベースに送信する」106 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「日付と時刻関数」142 ページ
- 「DATE データ型」112 ページ
- 「DATETIME データ型」113 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「timestamp_format オプション」『SQL Anywhere サーバー データベース管理』
- 「timestamp_with_time_zone_format オプション」『SQL Anywhere サーバー データベース管理』
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

- **Transact-SQL**　SMALLDATETIME は Adaptive Server Enterprise によってサポートされます。Adaptive Server Enterprise の SMALLDATETIME 型では、1900 年 1 月 1 日から 2079 年 6 月 6 日までの日付がサポートされ、値の時刻の部分でサポートされている精度はより低くなっています。SQL Anywhere では、SMALLDATETIME は、これらの制約なしに TIMESTAMP として実装されます。SQL Anywhere と Adaptive Server Enterprise の間でデータを移行する場合は、これらの違いに注意してください。

TIME データ型

TIME データ型は、時、分、秒、秒以下で構成される時刻を格納するために使用します。

構文

TIME

備考

アプリケーションが文字列として TIME 値を取得するときのフォーマットは、time_format オプションの設定によって決まります。たとえば、time_format オプション設定に従い、TIME 値 23:59:59.999999 は、23:59:59、23:59:59.999、または 23:59:59.999999 としてアプリケーションに返されます。

TIME 値には 8 バイトの記憶領域が必要です

ODBC を使用するときに、(ODBC TIME_STRUCT 構造体を使用して) バイナリ値として送信または取得される TIME 値は、時、分、秒の精度に制限されます。秒の小数部分は構造体には含まれません。そのため、精度を高めることを希望する場合は、TIME 値を文字列として送信または取得してください。

参照

- 「時間フォーマット」109 ページ
- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「日付と時刻関数」142 ページ
- 「DATE データ型」112 ページ
- 「DATETIME データ型」113 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「timestamp_format オプション」『SQL Anywhere サーバー データベース管理』
- 「timestamp_with_time_zone_format オプション」『SQL Anywhere サーバー データベース管理』
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　SQL/2008 と互換性があります。

- **Transact-SQL**　TIME データ型は Adaptive Server Enterprise によってサポートされます。ただし、Adaptive Server Enterprise では、マイクロ秒の精度 (6 桁) ではなく、ミリ秒の精度 (3 桁) がサポートされます。SQL Anywhere と Adaptive Server Enterprise の間でデータを移行する場合は、これらの違いに注意してください。TIME 値を移行するには、Adaptive Server Enterprise の BIGTIME データ型を使用します。

TIMESTAMP データ型

TIMESTAMP データ型は、年、月、日、時、分、秒、秒以下で構成される時刻を格納するために使用します。

構文

TIMESTAMP

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

TIMESTAMP 値は、年、月、日、時、分、秒、秒の小数部分で構成されます。秒以下は 6 桁まで格納されます。

アプリケーションが文字列として TIMESTAMP 値を取得するときのフォーマットは、timestamp_format オプションの設定によって決まります。たとえば、TIMESTAMP 値 2010/04/01T23:59:59.999999 は、timestamp_format オプション設定に従い、アプリケーションに 2010/04/01 23:59:59 または April 1, 2010 23:59:59.999999 として返されます。

TIMESTAMP 値には 8 バイトの記憶領域が必要です。

TIMESTAMP データ型では、DATE 型と同じく 0001 ～ 9999 年の日付を格納できますが、TIMESTAMP データ型の有効範囲は 1600-02-28 23:59:59 ～ 7911-01-01 00:00:00 です。TIMESTAMP 値では、この範囲より前または後の時間および分の部分は保持されません。

注意
TIMESTAMP 値の精度が下がると、分または秒に関連する組み込み関数が無効な結果を生成します。

TIMESTAMP 値を TIMESTAMP WITH TIME ZONE に変換する場合、接続の time_zone_adjustment 設定が結果のタイムゾーンオフセットに使用されます。つまり、値はその接続に「ローカル」なものであると見なされます。TIMESTAMP WITH TIME ZONE 値を TIMESTAMP に変換すると、オフセットは廃棄されます。

参照

- 「日付と時刻をデータベースに送信する」106 ページ
- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「日付と時刻関数」142 ページ
- 「DATE データ型」112 ページ
- 「DATETIME データ型」113 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「date_order オプション」『SQL Anywhere サーバー データベース管理』
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「nearest_century オプション」『SQL Anywhere サーバー データベース管理』
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「timestamp_format オプション」『SQL Anywhere サーバー データベース管理』
- 「timestamp_with_time_zone_format オプション」『SQL Anywhere サーバー データベース管理』
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008** SQL/2008 と互換性があります。
- **Transact-SQL** Adaptive Server Enterprise は、TIMESTAMP 値に対して DATETIME 型を使用します。

TIMESTAMP WITH TIME ZONE データ型

タイムゾーンオフセット付きの時刻を格納します。

構文

TIMESTAMP WITH TIME ZONE

備考

TIMESTAMP WITH TIME ZONE 値は、年、月、日、時、分、秒、秒の小数位、および協定世界時 (UTC: Coordinated Universal Time) との差異を示す分数を含んでいます。秒以下は 6 桁まで格納されます。

アプリケーションが文字列として TIMESTAMP WITH TIME ZONE 値を取得するときのフォーマットは、timestamp_with_time_zone_format オプションの設定によって決まります。たとえば、TIMESTAMP WITH TIME ZONE 値 2010/04/01T23:59:59.999999-6:00 は、timestamp_with_time_zone_format オプション設定に従い、アプリケーションに 2010/04/01 23:59:59 -06:00 または April 1, 2010 23:59:59.999999 -06:00 として返されます。

TIMESTAMP WITH TIME ZONE 値には 10 バイトの記憶領域が必要です。

TIMESTAMP WITH TIME ZONE データ型では、DATE 型と同じく 0001 ～ 9999 年の日付を格納できますが、TIMESTAMP WITH TIME ZONE データ型の有効範囲は 1600-02-28 23:59:59 ～ 7911-01-01 00:00:00 です。TIMESTAMP WITH TIME ZONE 値では、この範囲より前または後の時間および分の部分は保持されません。

計算カラムまたはマテリアライズドビューに TIMESTAMP WITH TIME ZONE を使用しないでください。時刻を制御する time_zone_adjustment オプションの値は、場所に基づく接続と日付によって異なるためです。

2 つの TIMESTAMP WITH TIME ZONE 値が UTC において同じ時刻を表す場合、適用されている TIME ZONE オフセットに関係なく同じ時刻と見なされます。たとえば、次の文では、結果が同じと見なされるため、**Yes** が返されます。

```
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
SELECT  'Yes'
ELSE
SELECT 'No'
END IF;
```

TIMESTAMP WITH TIME ZONE 値からタイムゾーンオフセットを省略すると、タイムスタンプが日付と時刻を標準時間または夏時間のいずれで表していても、クライアントの現在の UTC オフセットがデフォルト設定されます。たとえば、クライアントが東部標準タイムゾーンに属して

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

おり、夏時間が有効なときに次の文を実行した場合、大西洋標準タイムゾーン (UTC から -4 時間) に該当するタイムゾーンでタイムスタンプが返されます。

```
SELECT CAST('2009/01/30 12:34:55' AS TIMESTAMP WITH TIME ZONE);
```

- **TIMESTAMP WITH TIME ZONE と他のデータ型の比較**　TIMESTAMP WITH TIME ZONE 値をタイムゾーン付きではないタイムスタンプと比較することはおすすめしません。これは、クライアントのデフォルトのタイムゾーンオフセットは、クライアントの地理的な場所および日付によって異なるためです。

  クライアントの現在のタイムゾーンオフセットを分単位で判別するには、次の文を実行します。

  ```
  SELECT CONNECTION_PROPERTY( 'TimeZoneAdjustment' );
  ```

  注意
  TimeZoneAdjustment 接続プロパティは、Ultra Light データベースではサポートされません。

- **TIMESTAMP WITH TIME ZONE との変換**　TIMESTAMP 値を TIMESTAMP WITH TIME ZONE に変換する場合、接続の time_zone_adjustment 設定が結果のタイムゾーンオフセットに使用されます。つまり、値はその接続に「ローカル」なものであると見なされます。TIMESTAMP WITH TIME ZONE 値を TIMESTAMP に変換すると、オフセットは廃棄されます。文字列型、日付型、または時刻型以外の型との間の変換はサポートされていません。

参照

- 「日付と時刻の比較」130 ページ
- 「日付と時刻をデータベースに送信する」106 ページ
- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「日付と時刻関数」142 ページ
- 「DATE データ型」112 ページ
- 「DATETIME データ型」113 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「date_order オプション」『SQL Anywhere サーバー データベース管理』
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「nearest_century オプション」『SQL Anywhere サーバー データベース管理』
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「timestamp_format オプション」『SQL Anywhere サーバー データベース管理』
- 「timestamp_with_time_zone_format オプション」『SQL Anywhere サーバー データベース管理』
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　TIMESTAMP WITH TIME ZONE のサポートは、SQL/2008 標準のオプションの SQL 言語機能 F411 です。

バイナリデータ型

バイナリデータ型は、データベースに認識されないイメージやその他の種類の情報を含めたバイナリデータを格納するために使用します。

BINARY データ型

BINARY データ型は、指定した最大長 (バイト単位) のバイナリデータを格納するために使用します。

構文

BINARY [(*max-length*)]

パラメーター

- **max-length**　値のバイト単位での最大長。長さの範囲は 1 ～ 32767 です。指定しない場合の長さは 1 です。

備考

比較時に、BINARY 値はバイトごとに正確に比較されます。この比較方法は CHAR データ型とは異なります。CHAR の場合、データベースの照合順を使用して値が比較されます。1 つのバイナリ文字列がその他のプレフィクスの場合、短い文字列は長い文字列よりも文字数が少ないと見なされます。

CHAR 値とは異なり、文字セットの変換時に BINARY 値は変換されません。

セマンティック上、BINARY は VARBINARY と同じです。可変長型です。他のデータベース管理システムでは、BINARY は固定長型です。

参照

- 「VARBINARY データ型」124 ページ
- 「LONG BINARY データ型」123 ページ
- 「文字列関数」148 ページ
- 「ビット処理演算子」11 ページ

標準と互換性

- **SQL/2008**　BINARY データ型は、SQL/2008 標準の SQL 言語機能 T021 です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

IMAGE データ型

IMAGE データ型は、任意の長さのバイナリデータを格納するために使用します。

構文

IMAGE

備考

IMAGE は LONG BINARY として実装されるドメインです。

参照

- 「LONG BINARY データ型」123 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

LONG BINARY データ型

LONG BINARY データ型は、任意の長さのバイナリデータを格納するために使用します。

構文

LONG BINARY

備考

最大サイズは 2 GB から 1 バイトを差し引いた値です (2^{31} - 1)。

参照

- 「BINARY データ型」122 ページ
- 「VARBINARY データ型」124 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　LONG BINARY データ型は、SQL/2008 標準の SQL 言語機能 T021 "BINARY and VARBINARY data types" と T041 "Basic LOB data type support" によって構成されています。

UNIQUEIDENTIFIER データ型

UNIQUEIDENTIFIER データ型は、UUID (GUID とも呼ばれる) の値を格納するために使用します。

構文

UNIQUEIDENTIFIER

備考

UNIQUEIDENTIFIER データ型は、通常は、ローをユニークに識別する UUID (ユニバーサルユニーク識別子) 値を保持するために、プライマリキーまたはその他のユニークカラムに使用されます。NEWID 関数では、あるコンピューターで生成される UUID 値が他のコンピューターで生成される UUID と一致しないように UUID 値が生成されます。したがって、NEWID を使用して生成された UNIQUEIDENTIFIER 値は、同期環境でキーとして使用できます。

次に例を示します。

```
CREATE TABLE T1 (
   pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
   c1 INT );
```

UUID 値は、GUID (グローバルユニーク識別子) とも呼ばれます。UUID 値にはハイフンが含まれます。これは他の RDBMS と互換性を持たせるためです。デフォルト設定は、uuid_has_hyphens option を Off にして変更することができます。

SQL Anywhere は、必要に応じて UNIQUEIDENTIFIER 値を文字列値とバイナリ値の間で自動的に変換します。

UNIQUEIDENTIFIER 値は、BINARY(16) として格納されますが、クライアントアプリケーションには BINARY(36) として示されます。このため、クライアントが値を文字列としてフェッチした場合に、結果に対して十分な領域が割り付けられることが保証されます。ODBC クライアントアプリケーションでは、uniqueidentifier 値が SQL_GUID 型として示されます。

参照

- 「NEWID デフォルト」『SQL Anywhere サーバー SQL の使用法』
- 「NEWID 関数 [その他]」286 ページ
- 「UUIDTOSTR 関数 [文字列]」383 ページ
- 「STRTOUUID 関数 [文字列]」359 ページ
- 「uuid_has_hyphens オプション」『SQL Anywhere サーバー データベース管理』
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

VARBINARY データ型

VARBINARY データ型は、指定した最大長 (バイト単位) のバイナリデータを格納するために使用します。

構文

VARBINARY [(*max-length*)]

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **max-length**　値のバイト単位での最大長。長さの範囲は 1 ～ 32767 です。指定しない場合の長さは 1 です。

備考

比較時に、VARBINARY 値はバイトごとに正確に比較されます。この比較方法は CHAR データ型とは異なります。CHAR の場合、データベースの照合順を使用して値が比較されます。1 つのバイナリ文字列がその他のプレフィクスの場合、短い文字列は長い文字列よりも文字数が少ないと見なされます。

CHAR 値とは異なり、文字セットの変換時に VARBINARY 値は変換されません。

VARBINARY は BINARY VARYING と指定することもできます。どの構文を使用する場合でも、データ型は VARBINARY と記述されます。

参照

- 「BINARY データ型」122 ページ
- 「LONG BINARY データ型」123 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　VARBINARY データ型は、SQL/2008 標準の SQL 言語機能 T021 の "BINARY and VARBINARY data types" によって構成されています。

ドメイン

「ドメイン」は、適切な位置に精度や小数点以下の桁数を含み、さらにオプションとしてデフォルト値や CHECK 条件などを含んでいる、組み込みデータ型のエイリアスです。ドメインには、通貨データ型のように SQL Anywhere が事前に定義したものもありますが、ユーザーが独自のドメインを追加することもできます。

ドメインは「ユーザー定義データ型」とも呼ばれ、データベース全体を通して、カラムを同じ NULL または NOT NULL 条件、同じ DEFAULT 設定、同じ CHECK 条件を持つ同じデータ型に自動的に定義できます。ドメインはデータベース全体の一貫性を高め、特定の種類のエラーを防止するのに役立ちます。

簡単なドメイン

ドメインを作成するには、CREATE DOMAIN 文を使用します。

次の文は street_address という名前の、35 文字の文字列のデータ型を作成します。

```
CREATE DOMAIN street_address CHAR( 35 );
```

CREATE DATATYPE は CREATE DOMAIN の代わりに使用できますが、おすすめしません。

データ型を作成するには RESOURCE 権限が必要です。データ型を一度作成すると、CREATE DOMAIN 文を実行したユーザー ID がそのデータ型の所有者となります。このデータ型はすべ

てのユーザーが使用できます。他のデータベースオブジェクトとは異なり、所有者名をデータ型名の前に置きません。

カラムを定義する場合は、street_address データ型を他のデータ型とまったく同じように使用します。たとえば、2 つのカラムがある次のテーブルでは、2 番目のカラムが street_address カラムです。

```
CREATE TABLE twocol (
   id INT,
   street street_address
);
```

DROP DOMAIN 文を使って、その所有者または DBA 権限を持つユーザーがドメインを削除できます。

```
DROP DOMAIN street_address;
```

この文を実行できるのは、このデータ型をデータベースのどのテーブルでも使用していないときのみです。使用されているドメインを削除しようとすると、エラーメッセージが表示されいます。

参照

- 「CREATE DOMAIN 文」533 ページ

ドメインの制約とデフォルト

NULL 入力可や DEFAULT 値の設定など、カラムに関連する属性の多くをドメインに組み込むことができます。データ型上で定義されたカラムは、NULL 設定、CHECK 条件、DEFAULT 値を自動的に継承します。このため、データベース全体を通してカラムに同じような意味を持たせ、統一性をとることができます。

たとえば、SQL Anywhere サンプルデータベース内の多くのプライマリキーカラムは、ID 番号を保持する整数カラムです。次の文は、このようなカラムに有効なデータ型を作成します。

```
CREATE DOMAIN id INT
NOT NULL
DEFAULT AUTOINCREMENT
CHECK( @col > 0 );
```

デフォルトでは、id データ型を使用して作成されたカラムには NULL 値を格納できません。このため、これらのカラムにはデフォルト値として自動的に増分された値が設定され、必ず整数を保持しています。どのような識別子も *@col* 変数の中の *col* の代わりに使用できます。

カラムに対して明示的に属性を指定すると、データ型の属性を上書きできます。id データ型を使用して作成されたカラムに明示的に NULL が許可されると、id データ型の設定にかかわらず、NULL を使用できます。

互換性

- **名前を付けた制約とデフォルト**　SQL Anywhere では、ドメインをベースデータ型で作成し、オプションとして NULL または NOT NULL 条件、デフォルト値、CHECK 条件を含めます。名前付き制約と名前付きデフォルトはサポートしません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **データ型を作成する**　SQL Anywhere では、sp_addtype システムプロシージャーを使って、ドメインを追加したり、CREATE DOMAIN 文を使用したりできます。

データ型変換

型の変換は自動的に発生することもあれば、CAST または CONVERT 関数を使って明示的に型変換が要求されることもあります。次の関数を使うことによっても、強制的に型変換できます。

- **DATE 関数**　式を日付に変換し、時間、分、秒を削除します。変換エラーがあればレポートされます。
- **DATETIME 関数**　式を TIMESTAMP に変換し、タイムゾーンがあれば削除します。変換エラーがあればレポートされます。
- **STRING 関数**　この関数は、CAST(value AS LONG VARCHAR) と同じです。
- **VALUE+0.0**　CAST (value AS DECIMAL) と同じです。

自動データ型変換の概要を次に示します。

- 数値式の中、あるいは引数として数値が期待される関数への引数として文字列を使う場合は、文字列は数字に変換されます。
- 文字列式の中、あるいは文字列関数の引数として数字を使う場合は、数字は文字列に変換されてから使用されます。
- すべての日付定数は文字列として指定されます。文字列は、自動的に日付に変換されてから使用されます。

自動的なデータベース変換が適切ではない場合があります。たとえば、次の例では自動データ型変換が失敗します。

```
'12/31/90' + 5
'a' > 0
```

参照

- 「データ型変換関数」141 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「STRING 関数 [文字列]」359 ページ
- 「CAST 関数 [データ型変換]」165 ページ

データ型間の比較

データ型の異なる引数の間で比較 (= など) を行う場合、1 つのデータ型で比較操作ができるように、1 つ以上の引数を変換します。

規則によっては、変換に失敗したり、比較が予期しない結果になることがあります。そのような場合は、CAST または CONVERT を使用して引数の 1 つを明示的に変換してください。

引数を明示的に別のデータ型にキャストして、これらの変換規則を上書きすることができます。たとえば、DATE 型と CHAR 型の引数を CHAR 型として比較するには、DATE を CHAR に明示的にキャストしてください。

参照

- 「CAST 関数 [データ型変換]」165 ページ

損失を伴う変換および置換文字

変換後の文字セットで文字を表現できない場合は、代わりに置換文字が使用されます。この種の変換は、「損失を伴う」と考えられます。つまり、変換後の文字セットで元の文字を表現できない場合は、その文字が失われます。

また、文字セットによって置換文字が異なる場合があるだけでなく、ある文字セットの置換文字が別の文字セットでは非置換文字であることもあります。1 文字に対して複数の変換が実行されるときには、このことを理解することが重要になります。最終的な文字が、変換後の文字セットで期待される置換文字として表示されない可能性があるためです。

たとえば、クライアントの文字セットが Windows-1252 で、データベースの文字セットが ISO_8859-1:1987 (一部のバージョンの UNIX における米国のデフォルト) であるとします。次に、非 Unicode のクライアントアプリケーション (たとえば Embedded SQL) がユーロ記号を CHAR、VARCHAR、または LONG VARCHAR カラムに挿入しようとします。この文字は CHAR 文字セットに存在しないため、ISO_8859-1:1987 の置換文字である 0x1A が挿入されます。

同じ ISO_8859-1:1987 置換文字が Unicode としてフェッチされる (たとえば ODBC で SELECT * FROM t を SQL_C_WCHAR にバインドされたカラムにフェッチする) 場合、この文字は Unicode のコードポイント U+001A になります (Unicode では、コードポイント U+001A はレコードセパレーター制御文字です)。ただし、Unicode の置換文字はコードポイント U+FFFD です。この例は、データに置換文字が含まれる場合であっても、複数回の変換が原因で、これらの文字が変換後の文字セットの置換文字に変換されない可能性があることを示しています。

このため、複数の文字セット間で変換を行う場合は、置換文字がどのように使用されるかを理解してテストすることが重要です。

on_charset_conversion_failure オプションを使用すると、文字を変換後の文字セットで表現できない場合の変換時の動作を指定しやすくなることがあります。

参照

- 「データ型変換」127 ページ
- 「CHAR と NCHAR の比較」129 ページ
- 「on_charset_conversion_failure オプション」『SQL Anywhere サーバー データベース管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CHAR と NCHAR の比較

CHAR 型 (CHAR、VARCHAR、LONG VARCHAR) の値と NCHAR 型 (NCHAR、NVARCHAR、LONG NVARCHAR) の値との間で比較を実行する場合は、比較を実行する型を判断するために推定規則が使用されます。一般的に、一方の値のみがカラム参照に基づいている場合は、カラム参照が含まれている値の型を使用して比較が実行されます。

推定規則では、値がカラム参照に基づいているかどうかが重要です。一方の値が変数、ホスト変数、またはリテラル定数であるか、またはカラム参照に基づいていない複雑な式であり、かつもう一方の値がカラム参照に基づいている場合は、暗黙的に、定数に基づく値がカラムに基づく値にキャストされます。

次に、推定規則を適用順に示します。

- NCHAR 値がカラム参照に基づいている場合は、暗黙的に CHAR 値が NCHAR にキャストされ、NCHAR として比較されます。NCHAR 値と CHAR 値の両方がカラム参照に基づいている場合も同様です。

- NCHAR 値がカラム参照に基づいておらず、かつ CHAR 値がカラム参照に基づいている場合は、暗黙的に NCHAR 値が CHAR にキャストされ、CHAR として比較されます。

  NCHAR から CHAR への変換が予想される場合は、on_charset_conversion_failure オプションの設定を検討することが重要です。このオプションでは、NCHAR 文字を CHAR 文字セットで表せないときの動作を制御するからです。

- どちらの値もカラム参照に基づいていない場合、暗黙的に CHAR 値が NCHAR にキャストされ、NCHAR として比較されます。

例

条件 Employees.GivenName = N'Susan' では、CHAR カラム (Employees.GivenName) をリテラル N'Susan' と比較します。値 N'Susan' は CHAR にキャストされるため、この比較は次のように記述された場合と同等に行われます。

```
Employees.GivenName  = CAST( N'Susan' AS CHAR )
```

または、条件 Employees.GivenName = T.nchar_column では、値 T.nchar_column を CHAR にキャストできないことが検出されます。この比較は、次のように記述された場合と同等に実行され、Employees.GivenName のインデックスは使用できません。

```
CAST( Employees.GivenName AS NCHAR ) = T.nchar_column;
```

参照

- 「NCHAR から CHAR への変換」133 ページ
- 「NCHAR から CHAR への変換」133 ページ
- 「損失を伴う変換および置換文字」128 ページ
- 「CAST 関数 [データ型変換]」165 ページ
- 「CONVERT 関数 [データ型変換]」179 ページ
- 「CAST 関数 [データ型変換]」165 ページ
- 「on_charset_conversion_failure オプション」『SQL Anywhere サーバー データベース管理』

数値データ型間の比較

SQL Anywhere では、数値データ型の比較に次の規則を使用します。規則は以下の表示順で検証され、一致する最初の規則が使用されます。

1. 一方の引数が TINYINT 型で、もう一方の引数が INTEGER 型の場合は、両方を INTEGER に変換してから比較を行う。

2. 一方の引数が TINYINT 型で、もう一方の引数が SMALLINT 型の場合は、両方を SMALLINT に変換してから比較を行う。

3. 一方の引数が UNSIGNED SMALLINT 型で、もう一方の引数が INTEGER 型の場合は、両方を INTEGER に変換してから比較を行う。

4. 引数のデータ型に共通のスーパータイプがある場合、その共通のスーパータイプデータ型に変換してから比較を行う。スーパータイプは、次の各リストの最終的なデータ型です。

   ●BIT » TINYINT » UNSIGNED SMALLINT » UNSIGNED INTEGER » UNSIGNED BIGINT » NUMERIC

   ●SMALLINT » INTEGER » BIGINT » NUMERIC

   ●REAL » DOUBLE

   ●CHAR » LONG VARCHAR

   ●BINARY » LONG BINARY

   たとえば、2 つの引数が BIT 型と TINYINT 型である場合、これらの引数は NUMERIC 型に変換されます。

日付と時刻の比較

次の表に、特定のデータ型を日付、時刻、または日付／時刻のデータ型と比較したときの暗黙の変換をまとめます。

データ型	データ型	変換
CHAR	DATE	CHAR は TIMESTAMP にキャストされ、DATE は TIMESTAMP にキャストされる
CHAR	TIME	CHAR は TIME にキャストされる

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

データ型	データ型	変換
CHAR	TIMESTAMP	CHAR は TIMESTAMP にキャストされる
CHAR	TIMESTAMP WITH TIME ZONE	CHAR は TIMESTAMP WITH TIME ZONE にキャストされる
DATE	TIME	無効
DATE	TIMESTAMP	DATE は TIMESTAMP にキャストされる
DATE	TIMESTAMP WITH TIME ZONE	DATE は TIMESTAMP WITH TIME ZONE にキャストされる
DATE	SMALLINT、INTEGER、BIGINT、NUMERIC	SMALLINT、INTEGER、BIGINT、NUMERIC の値は日付文字列として扱われ、TIMESTAMP にキャストされる。DATE は TIMESTAMP にキャストされる
DATE	REAL、FLOAT、DOUBLE	REAL、FLOAT、DOUBLE は 0000-02-29 から経過した日数として扱われ、TIMESTAMP にキャストされる。DATE は TIMESTAMP にキャストされる
TIME	TIMESTAMP	TIMESTAMP は TIME にキャストされる
TIME	TIMESTAMP WITH TIME ZONE	無効
TIMESTAMP	TIMESTAMP WITH TIME ZONE	TIMESTAMP は TIMESTAMP WITH TIME ZONE にキャストされる
TIMESTAMP	SMALLINT、INTEGER、BIGINT、NUMERIC	SMALLINT、INTEGER、BIGINT、NUMERIC の値は日付文字列として扱われ、TIMESTAMP にキャストされる

データ型	データ型	変換
TIMESTAMP	REAL、FLOAT、DOUBLE	REAL、FLOAT、DOUBLE は 0000-02-29 から経過した日数として扱われ、TIMESTAMP にキャストされる

これ以降では、前述の表で示された情報をさらに詳しく説明しています。

1. データ型 TIME の値に比較できるのは、データ型 TIME、TIMESTAMP、CHAR の値だけです。それ以外のデータ型の値との比較は、変換エラーになります。時間値と別のデータ型の値を比較するときには、比較データ型は TIME になります。

2. TIMESTAMP、SMALLINT、INTEGER、BIGINT、NUMERIC、REAL、FLOAT、DOUBLE の値を DATE の値と比較するとき、比較データ型は常に TIMESTAMP になります。

3. TIMESTAMP WITH TIME ZONE 値を DATE 値と比較するとき、比較データ型は TIMESTAMP WITH TIME ZONE になります。

4. 時間値が TIMESTAMP にキャストされると、結果は、現在の日付と時間値を組み合わせたかたちになります。

5. データ型 SMALLINT、INTEGER、BIGINT、NUMERIC の正確な数値は、日付値に変換できます。変換は、数字を文字列として扱うことで実行されます。たとえば、整数値 20100401 は 2010 年 4 月 1 日を表します。

6. 符号なしの正確な数値データ型である BIT、TINYINT、UNSIGNED SMALLINT、UNSIGNED INTEGER、UNSIGNED BIGINT は、日付値には変換できません。

7. 概数値データ型である REAL、FLOAT、DOUBLE は、架空の日付 0000-02-29 から経過した日数として数値を扱うことで日付に変換できます。たとえば、307 は 0001-01-01 を表し、734169 は 2010-04-01 を表します。

参照

- 「日付と時刻データ型」105 ページ

Transact-SQL の文字列から日付／時刻への変換

時刻値のみ (日付なし) を含む文字を日付／時刻データ型に変換すると、SQL Anywhere では現在の日付が使用されます。

時刻の部分が 3 桁未満の場合、ピリオドまたはコロンが後に付いているかどうかにかかわらず、SQL Anywhere では同様に解釈されます。つまり、1 桁は 10 単位、2 桁は 100 単位、3 桁は 1000 単位です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

SQL Anywhere では、セパレーターの有無にかかわらず、同じ方法でミリ秒値が変換されます。

12:34:56.7 は 12:34:56.700 に
12:34:56:7 は 12:34:56.700 に
12.34.56.78 は 12:34:56.780 に
12.34.56:78 は 12:34:56.780 に
12:34:56.789 は 12:34:56.789 に
12:34:56:789 は 12:34:56.789 に

その他の比較

1. CHAR (CHAR、VARCHAR、LONG VARCHAR など、ただし NCHAR 型以外) が混在しているデータ型の場合は、LONG VARCHAR に変換してから比較を行う。
2. いずれかの引数のデータ型が UNIQUEIDENTIFIER の場合は、UNIQUEIDENTIFIER に変換してから比較を行う。
3. いずれかの引数のデータ型がビット配列 (VARBIT または LONG VARBIT) の場合は、LONG VARBIT に変換してから比較を行う。
4. 一方の引数が CHARACTER データ型で、もう一方の引数が BINARY データ型の場合は、BINARY に変換してから比較を行う。
5. 一方の引数が CHAR 型で、もう一方の引数が NCHAR 型の場合は、事前に定義された推定規則を使用する。
6. 規則が存在しない場合は、NUMERIC 型に変換してから比較を行う。

   たとえば、2 つの引数が REAL 型と CHAR 型の場合、これらの引数はいずれも NUMERIC 型に変換されます。

参照

● 「CHAR と NCHAR の比較」 129 ページ

NCHAR から CHAR への変換

NCHAR から CHAR への変換は、CHAR データと NCHAR データの比較の一環として、または具体的に要求されたときに起こることがあります。CHAR 文字セットによって CHAR 型に変換できない NCHAR 文字がある可能性があるため、このタイプの変換は損失を伴います。NCHAR 文字が CHAR に変換できない場合は、代わりに CHAR 文字セットの置換文字が使用されます。シングルバイト文字セットの場合、一般的に 16 進の 1A です。

on_charset_conversion_failure オプションの設定に応じて、文字が変換できないときに次のいずれかが起こります。

●置換文字が使用され、警告が発行されない

●置換文字が使用され、警告が発行される

●エラーが返される

そのため、NCHAR から CHAR への変換時はこのオプションを検討することが重要です。

参照

- 「CHAR と NCHAR の比較」129 ページ
- 「on_charset_conversion_failure オプション」『SQL Anywhere サーバー データベース管理』

NUMERIC 型と文字列型への NULL 定数の変換

NULL 定数を NUMERIC または文字列型 (CHAR、VARCHAR、LONG VARCHAR、BINARY、VARBINARY、LONG BINARY) に変換すると、サイズは 0 に設定されます。次に例を示します。

SELECT CAST(NULL AS CHAR) は CHAR(0) を返します。

SELECT CAST(NULL AS NUMERIC) は NUMERIC(1,0) を返します。

ビット配列の変換

整数をビット配列に変換する

整数をビット配列に変換すると、ビット配列の長さは整数型のビット数になり、ビット配列の値はバイナリ表現になります。整数の最上位ビットは配列の最初のビットになります。

例

SELECT CAST(CAST(1 AS BIT) AS VARBIT) は 1 を含む VARBIT(1) を返します。

SELECT CAST(CAST(8 AS TINYINT) AS VARBIT) は 00001000 を含む VARBIT(8) を返します。

SELECT CAST(CAST(194 AS INTEGER) AS VARBIT) は 00000000000000000000000011000010 を含む VARBIT(32) を返します。

バイナリをビット配列に変換する

長さ *n* のバイナリ型をビット配列に変換すると、配列の長さは *n* * 8 ビットになります。ビット配列の最初の 8 ビットはバイナリ値の最初のバイトになります。バイナリ値の最上位ビットは配列の最初のビットになります。ビット配列の次の 8 ビットはバイナリ値の 2 番目のバイトになります。以降も同様です。

例

SELECT CAST(0x8181 AS VARBIT) は 1000000110000001 を含む VARBIT(16) を返します。

文字をビット配列に変換する

長さ *n* の文字データ型をビット配列に変換すると、配列の長さは *n* ビットになります。各文字は '0' または '1' で、配列の対応するビットには値 0 または 1 が割り当てられます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

SELECT CAST('001100' AS VARBIT) は 001100 を含む VARBIT(6) を返します。

ビット配列を整数に変換する

ビット配列を整数データ型に変換すると、ビット配列のバイナリ値は、最初に最上位ビットを使用し、整数型の格納形式に従って解釈されます。

例

SELECT CAST(CAST('11000010' AS VARBIT) AS INTEGER) は 194 (11000010_2 = 0xC2 = 194) を返します。

ビット配列をバイナリに変換する

ビット配列をバイナリに変換すると、配列の最初の 8 ビットがバイナリ値の最初のバイトになります。配列の最初のビットは、バイナリ値の最上位ビットになります。続く 8 ビットは 2 番目のバイトとして使用されます。以降も同様です。ビット配列の長さが 8 の倍数ではない場合、バイナリ値の最終バイトの最下位ビットに入力するときに追加の 0 が使用されます。

例

SELECT CAST(CAST('1111' AS VARBIT) AS BINARY) は 0xF0 (1111_2 は 11110000_2 = 0xF0 になります) を返します。

SELECT CAST(CAST('0011000000110001' AS VARBIT) AS BINARY) は 0x3031 (0011000000110001_2 = 0x3031) を返します。

ビット配列を文字に変換する

長さ *n* ビットのビット配列を文字データ型に変換すると、結果の長さは *n* 文字になります。結果の各文字には、配列のビットに応じて '0' または '1' が指定されます。

例

SELECT CAST(CAST('01110' AS VARBIT) AS VARCHAR) は文字列 '01110' を返します。

数値セットの変換

DOUBLE 型を NUMERIC 型に変換すると、上位 15 桁の精度が維持されます。

参照

- 「CAST 関数 [データ型変換]」 165 ページ
- 「CONVERT 関数 [データ型変換]」 179 ページ

Java と SQL のデータ型変換

Java 型と SQL 型の間のデータ型変換は、Java ストアドプロシージャーと JDBC アプリケーションの両方で必要です。Java から SQL、SQL から Java へのデータ型変換は、JDBC 標準に準拠して行われます。次の表で変換について説明します。

Java から SQL のデータ型変換

Java 型	SQL 型
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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Java 型	SQL 型
java.lang.Float	REAL
java.lang.Integer	INTEGER
java.lang.Long	BIGINT

SQL から Java のデータ型変換

SQL 型	Java 型
CHAR	String
VARCHAR	String
TEXT	String
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

SQL 型	Java 型
BINARY	byte[ ]
VARBINARY	byte[ ]
LONG BINARY	byte[ ]
IMAGE	byte[ ]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

空間データ型

SQL Anywhere では多くの空間データ型がサポートされます。これらのデータ型のマニュアルは、空間 SQL API のマニュアルと同じ場所にあります。

参照

- 「サポートされる空間データ型とその階層」『SQL Anywhere サーバー 空間データサポート』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL 関数

関数は、データベースの情報を返すために使用します。関数は、式を使用できるところではどこからでも呼び出すことができます。

マニュアルに特に指定がないかぎり、関数の引数のいずれかが NULL である場合は、NULL が返されます。

引数を省略できる場合、引数として DEFAULT を指定できます。

関数には、SQL 文と同じ構文の表記規則を使用します。

参照

- 「SQL 構文の表記規則」404 ページ

関数のタイプ

この項では、使用可能な関数をタイプ別に分類します。

参照

- 「空間データ用の関数」『Ultra Light データベース管理とリファレンス』

集合関数

集合関数は、データベースから選択されたローグループのデータを要約します。グループは、SELECT 文の GROUP BY 句を使用して形成されます。集合関数は、SELECT リストと、SELECT 文の HAVING 句と ORDER BY 句でのみ使用できます。

関数のリスト

次の集合関数を使用できます。

- 「AVG 関数 [集合]」
- 「BIT_AND 関数 [集合]」
- 「BIT_OR 関数 [集合]」
- 「BIT_XOR 関数 [集合]」
- 「COVAR_POP 関数 [集合]」
- 「COVAR_SAMP 関数 [集合]」
- 「COUNT 関数 [集合]」
- 「COUNT_BIG 関数 [集合]」
- 「CORR 関数 [集合]」
- 「FIRST_VALUE 関数 [集合]」
- 「GROUPING 関数 [集合]」
- 「LAST_VALUE 関数 [集合]」
- 「LIST 関数 [集合]」
- 「MAX 関数 [集合]」
- 「MEDIAN 関数 [集合]」
- 「MIN 関数 [集合]」
- 「REGR_AVGX 関数 [集合]」
- 「REGR_AVGY 関数 [集合]」
- 「REGR_COUNT 関数 [集合]」
- 「REGR_INTERCEPT 関数 [集合]」
- 「REGR_R2 関数 [集合]」
- 「REGR_SLOPE 関数 [集合]」
- 「REGR_SXX 関数 [集合]」
- 「REGR_SXY 関数 [集合]」
- 「REGR_SYY 関数 [集合]」
- 「SET_BITS 関数 [集合]」
- 「STDDEV 関数 [集合]」
- 「STDDEV_POP 関数 [集合]」
- 「STDDEV_SAMP 関数 [集合]」
- 「SUM 関数 [集合]」
- 「VAR_POP 関数 [集合]」
- 「VAR_SAMP 関数 [集合]」
- 「VARIANCE 関数 [集合]」
- 「XMLAGG 関数 [集合]」

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ビット配列関数

ビット配列関数を使用すると、ビット配列でタスクを実行できます。次のビット配列関数を使用できます。

- 「BIT_AND 関数 [集合]」
- 「BIT_OR 関数 [集合]」
- 「BIT_XOR 関数 [集合]」
- 「BIT_LENGTH 関数 [ビット配列]」
- 「BIT_SUBSTR 関数 [ビット配列]」
- 「COUNT_SET_BITS 関数 [ビット配列]」
- 「GET_BIT 関数 [ビット配列]」
- 「SET_BIT 関数 [ビット配列]」
- 「SET_BITS 関数 [集合]」

参照

- 「ビット処理演算子」 11 ページ
- 「sa_get_bits システムプロシージャー」 1086 ページ

ランキング関数

ランキング関数により、クエリで指定した順番に基づいて、結果セットにある各ローのランク値を計算できます。

- 「CUME_DIST 関数 [ランキング]」
- 「DENSE_RANK 関数 [ランキング]」
- 「PERCENT_RANK 関数 [ランキング]」
- 「RANK 関数 [ランキング]」

データ型変換関数

データ型変換関数を使用して、引数のデータ型を他のデータ型に変換したり、変換が可能かどうかをテストしたりします。

関数のリスト

次のデータ型変換関数を使用できます。

- 「CAST 関数 [データ型変換]」
- 「CONVERT 関数 [データ型変換]」
- 「HEXTOINT 関数 [データ型変換]」
- 「INTTOHEX 関数 [データ型変換]」
- 「ISDATE 関数 [データ型変換]」
- 「ISNUMERIC 関数 [その他]」
- 「TREAT 関数 [データ型変換]」

日付と時刻関数

日付と時刻の関数は、DATE、TIME、TIMESTAMP、および TIMESTAMP WITH TIME ZONE の各データ型に対する操作を実行します。

SQL Anywhere には、Transact-SQL の日付型と時間型に対する互換性サポート (DATETIME と SMALLDATETIME を含む) があります。これらの Transact-SQL データ型は、SQL Anywhere のネイティブな TIMESTAMP データ型を介したドメインとして実装されます。

日付の単位の指定

日付関数の多くは、「日付の単位」で構成される日付を使用します。次の表は、使用できる日付の単位の値を示します。

日付と時刻の関数を使用する場合、日付または時刻の減算にマイナス記号を指定できます。たとえば、31 日前のタイムスタンプを取得するには、次の文を実行します。

```
SELECT DATEADD(day, -31, NOW());
```

日付の単位	省略形	値
Year	yy	1-9999
Quarter	qq	1-4
Month	mm	1-12
Week	wk	1-54。日曜日を週の最初の日とします。54 週ある年は、土曜日から始まる閏年で発生します。
Day	dd	1-31
Dayofyear	dy	1-366
Weekday	dw	1 ～ 7 (日曜日 = 1、...、土曜日 = 7)
Hour	hh	0-23
Minute	mi	0-59
Second	ss	0-59
Millisecond	ms	0-999
Microsecond	mcs または us	0-999999

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

日付の単位	省略形	値
Calyearofweek	cyr	1-9999。その週が何年に開始したかを示します。週に年の最初の数日が含まれている場合は、その年の最初の曜日に応じて、週の最初の日が前年になる場合があります。年の最初の曜日が月曜日～木曜日の場合は、前年に属する日がその年に含まれることはありませんが、年の最初の曜日が金曜日～日曜日の場合、年の最初の週はその年の最初の月曜日から開始します。
Calweekofyear	cwk	1-53。指定した日付がその年の第何週であるかを示します。 ISO の週のシステムと ISO 8601 の日付と時刻の標準の詳細については、http://en.wikipedia.org/wiki/ISO_week_date を参照してください。
Caldayofweek	cdw	1-7。(月曜日 = 1、...、日曜日 = 7)
TZOffset	tz	-840 ～ 840

日付と時刻の関数のリスト

次の日付と時刻関数を使用できます。

- 「DATE 関数 [日付と時刻]」
- 「DATEADD 関数 [日付と時刻]」
- 「DATEDIFF 関数 [日付と時刻]」
- 「DATEFORMAT 関数 [日付と時刻]」
- 「DATENAME 関数 [日付と時刻]」
- 「DATEPART 関数 [日付と時刻]」
- 「DATETIME 関数 [日付と時刻]」
- 「DAY 関数 [日付と時刻]」
- 「DAYNAME 関数 [日付と時刻]」
- 「DAYS 関数 [日付と時刻]」
- 「DOW 関数 [日付と時刻]」
- 「GETDATE 関数 [日付と時刻]」
- 「HOUR 関数 [日付と時刻]」
- 「HOURS 関数 [日付と時刻]」
- 「MINUTE 関数 [日付と時刻]」
- 「MINUTES 関数 [日付と時刻]」
- 「MONTH 関数 [日付と時刻]」
- 「MONTHNAME 関数 [日付と時刻]」
- 「MONTHS 関数 [日付と時刻]」
- 「NOW 関数 [日付と時刻]」
- 「QUARTER 関数 [日付と時刻]」
- 「SECOND 関数 [日付と時刻]」
- 「SECONDS 関数 [日付と時刻]」
- 「SWITCHOFFSET 関数 [日付と時刻]」
- 「SYSDATETIMEOFFSET 関数 [日付と時刻]」
- 「TODAY 関数 [日付と時刻]」
- 「TODATETIMEOFFSET 関数 [日付と時刻]」
- 「WEEKS 関数 [日付と時刻]」
- 「YEAR 関数 [日付と時刻]」
- 「YEARS 関数 [日付と時刻]」
- 「YMD 関数 [日付と時刻]」

参照

- 「日付と時刻データ型」105 ページ
- 「Ultra Light のデータ型」『Ultra Light データベース管理とリファレンス』

ユーザー定義関数

ユーザー定義関数 (UDF) は、プログラムまたは環境のユーザーによって作成される関数です。ユーザー定義関数には、プログラムまたは環境に組み込まれている関数との相違点があります。

SQL Anywhere には、ユーザー定義関数を作成する 2 つのメカニズムがあります。SQL 言語を使用して関数を記述するか、Java を使用できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL のユーザー定義関数

CREATE FUNCTION 文を使用して、独自の SQL 関数を実装できます。

CREATE FUNCTION 文中の RETURN 文によって、関数のデータ型が決まります。

一度 SQL ユーザー定義関数を作成すると、同じデータ型の組み込み関数が使用される任意の場所でその関数を使用できます。

Java のユーザー定義関数

Java クラスを使用すると、必要に応じて関数をデータベースサーバーからクライアントアプリケーションに移動できるという追加の利点があるため、より強力で柔軟性に富んだユーザー定義関数を実装できます。

インストールされた Java クラスの「クラスメソッド」は、同じデータ型の組み込み関数が使用されるところでは必ずユーザー定義関数として使用できます。

インスタンスメソッドは、クラスの特定のインスタンスと関連しているため、標準のユーザー定義関数とは動作が異なります。

ユーザー定義関数とユーザー定義プロシージャーのどちらを作成するかの判断

関数はプロシージャーと似ています。関数とプロシージャーのどちらを作成するかは、必要な戻り値と呼び出すオブジェクトに基づいて判断します。判断する際には、以下に示すユニークな特性について検討します。

関数：

- 任意の型の単一値を返すことができ、RETURNS 句を使用して戻り値の型を宣言できる
- 式の使用が可能なほとんどの場所で使用できる
- 定義できるのは IN パラメーターのみ

プロシージャー：

- INOUT または OUT パラメーターを使用して複数の値を返すことができる
- 結果セットを返すことができる
- クエリの FROM 句で参照したり、CALL 文や Transact-SQL の EXECUTE 文を使用して参照できる
- 名前付きパラメーターを使用して呼び出すことができる

参照

- 「CREATE FUNCTION 文」563 ページ
- 「クラスの作成」『SQL Anywhere サーバー プログラミング』
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』

その他の関数

その他の関数は、算術式、文字列式、日付／時刻式、他の関数の戻り値に対して操作を実行します。

関数のリスト

次の各種関数を使用できます。

- 「ARGN 関数 [その他]」
- 「COALESCE 関数 [その他]」
- 「CONFLICT 関数 [その他]」
- 「ERRORMSG 関数 [その他]」
- 「ESTIMATE 関数 [その他]」
- 「ESTIMATE_SOURCE 関数 [その他]」
- 「EXPERIENCE_ESTIMATE 関数 [その他]」
- 「EXPLANATION 関数 [その他]」
- 「EXPRTYPE 関数 [その他]」
- 「GET_IDENTITY 関数 [その他]」
- 「GRAPHICAL_PLAN 関数 [その他]」
- 「GREATER 関数 [その他]」
- 「IDENTITY 関数 [その他]」
- 「IFNULL 関数 [その他]」
- 「INDEX_ESTIMATE 関数 [その他]」
- 「ISNULL 関数 [その他]」
- 「LESSER 関数 [その他]」
- 「NEWID 関数 [その他]」
- 「NULLIF 関数 [その他]」
- 「NUMBER 関数 [その他]」
- 「PLAN 関数 [その他]」
- 「REWRITE 関数 [その他]」
- 「ROW_NUMBER 関数 [その他]」
- 「SQLDIALECT 関数 [その他]」
- 「SQLFLAGGER 関数 [その他]」
- 「TRACEBACK 関数 [その他]」
- 「TRANSACTSQL 関数 [その他]」
- 「VAREXISTS 関数 [その他]」
- 「WATCOMSQL 関数 [その他]」

数値関数

数値関数は、数値データ型の算術演算を実行したり、数値情報を返したりします。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関数のリスト

次の数値関数を使用できます。

- 「ABS 関数 [数値]」
- 「ACOS 関数 [数値]」
- 「ASIN 関数 [数値]」
- 「ATAN 関数 [数値]」
- 「ATAN2 関数 [数値]」
- 「CEILING 関数 [数値]」
- 「COS 関数 [数値]」
- 「COT 関数 [数値]」
- 「DEGREES 関数 [数値]」
- 「EXP 関数 [数値]」
- 「FLOOR 関数 [数値]」
- 「LOG 関数 [数値]」
- 「LOG10 関数 [数値]」
- 「MOD 関数 [数値]」
- 「PI 関数 [数値]」
- 「POWER 関数 [数値]」
- 「RADIANS 関数 [数値]」
- 「RAND 関数 [数値]」
- 「REMAINDER 関数 [数値]」
- 「ROUND 関数 [数値]」
- 「SIGN 関数 [数値]」
- 「SIN 関数 [数値]」
- 「SQRT 関数 [数値]」
- 「TAN 関数 [数値]」
- 「TRUNCNUM 関数 [数値]」

Web サービス関数

HTTP 関数は、Web サービス内の HTTP 要求の処理を支援します。同様に、SOAP 関数は、Web サービス内の SOAP 要求の処理を支援します。

次の関数を使用できます。

- 「HTML_DECODE 関数 [その他]」
- 「HTML_ENCODE 関数 [その他]」
- 「HTTP_BODY 関数 [Web サービス]」
- 「HTTP_DECODE 関数 [Web サービス]」
- 「HTTP_ENCODE 関数 [Web サービス]」
- 「HTTP_HEADER 関数 [Web サービス]」
- 「HTTP_RESPONSE_HEADER 関数 [Web サービス]」
- 「HTTP_VARIABLE 関数 [Web サービス]」
- 「NEXT_HTTP_HEADER 関数 [Web サービス]」
- 「NEXT_HTTP_RESPONSE_HEADER 関数 [Web サービス]」
- 「NEXT_HTTP_VARIABLE 関数 [Web サービス]」
- 「NEXT_SOAP_HEADER 関数 [SOAP]」
- 「SOAP_HEADER 関数 [SOAP]」

また、Web サービスで使用できるシステムプロシージャーもあります。

参照

- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「Web サービスシステムプロシージャー」1030 ページ

文字列関数

文字列関数は、文字列に対して変換、抽出、操作の演算を実行したり、文字列に関する情報を返したりします。

マルチバイト文字セットを操作する場合は、使用する関数が文字とバイトのどちらの情報を返すかを十分に確認してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関数のリスト

次の文字列関数を使用できます。

- 「ASCII 関数 [文字列]」
- 「BASE64_DECODE 関数 [文字列]」
- 「BASE64_ENCODE 関数 [文字列]」
- 「BYTE_LENGTH 関数 [文字列]」
- 「BYTE_SUBSTR 関数 [文字列]」
- 「CHAR 関数 [文字列]」
- 「CHARINDEX 関数 [文字列]」
- 「CHAR_LENGTH 関数 [文字列]」
- 「COMPARE 関数 [文字列]」
- 「COMPRESS 関数 [文字列]」
- 「CSCONVERT 関数 [文字列]」
- 「DECOMPRESS 関数 [文字列]」
- 「DECRYPT 関数 [文字列]」
- 「DIFFERENCE 関数 [文字列]」
- 「ENCRYPT 関数 [文字列]」
- 「HASH 関数 [文字列]」
- 「INSERTSTR 関数 [文字列]」
- 「LCASE 関数 [文字列]」
- 「LEFT 関数 [文字列]」
- 「LENGTH 関数 [文字列]」
- 「LOCATE 関数 [文字列]」
- 「LOWER 関数 [文字列]」
- 「LTRIM 関数 [文字列]」
- 「NCHAR 関数 [文字列]」
- 「PATINDEX 関数 [文字列]」
- 「READ_CLIENT_FILE 関数 [文字列]」
- 「REGEXP_SUBSTR 関数 [文字列]」
- 「REPEAT 関数 [文字列]」
- 「REPLACE 関数 [文字列]」
- 「REPLICATE 関数 [文字列]」
- 「REVERSE 関数 [文字列]」
- 「RIGHT 関数 [文字列]」
- 「RTRIM 関数 [文字列]」
- 「SIMILAR 関数 [文字列]」
- 「SORTKEY 関数 [文字列]」
- 「SOUNDEX 関数 [文字列]」
- 「SPACE 関数 [文字列]」
- 「STR 関数 [文字列]」
- 「STRING 関数 [文字列]」
- 「STRTOUUID 関数 [文字列]」
- 「STUFF 関数 [文字列]」
- 「SUBSTRING 関数 [文字列]」
- 「TO_CHAR 関数 [文字列]」
- 「TO_NCHAR 関数 [文字列]」
- 「TRIM 関数 [文字列]」

- 「UCASE 関数 [文字列]」
- 「UNICODE 関数 [文字列]」
- 「UNISTR 関数 [文字列]」
- 「UPPER 関数 [文字列]」
- 「UUIDTOSTR 関数 [文字列]」
- 「XMLCONCAT 関数 [文字列]」
- 「XMLELEMENT 関数 [文字列]」
- 「XMLFOREST 関数 [文字列]」
- 「XMLGEN 関数 [文字列]」

システム関数

システム関数は、システム情報を返します。

関数のリスト

次のシステム関数を使用できます。

- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」
- 「CONNECTION_PROPERTY 関数 [システム]」
- 「DATALENGTH 関数 [システム]」
- 「DB_ID 関数 [システム]」
- 「DB_NAME 関数 [システム]」
- 「DB_EXTENDED_PROPERTY 関数 [システム]」
- 「DB_PROPERTY 関数 [システム]」
- 「EVENT_CONDITION 関数 [システム]」
- 「EVENT_CONDITION_NAME 関数 [システム]」
- 「EVENT_PARAMETER 関数 [システム]」
- 「NEXT_CONNECTION 関数 [システム]」
- 「NEXT_DATABASE 関数 [システム]」
- 「PROPERTY 関数 [システム]」
- 「PROPERTY_DESCRIPTION 関数 [システム]」
- 「PROPERTY_NAME 関数 [システム]」
- 「PROPERTY_NUMBER 関数 [システム]」
- 「SUSER_ID 関数 [システム]」
- 「SUSER_NAME 関数 [システム]」
- 「TSEQUAL 関数 [システム] (旧式)」
- 「USER_ID 関数 [システム]」
- 「USER_NAME 関数 [システム]」

注意

- 一部のシステム関数は、ストアドプロシージャーとして SQL Anywhere に実装されます。

- db_id、db_name、datalength は、組み込み関数として実装されます。

次の表は、実装されたシステム関数を示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

システム関数	説明
COL_LENGTH(*table-name*, *column-name* **)**	定義されたカラムの長さを返す
COL_NAME(*table-id*, *column-id* [, *database-id*] **)**	カラムの名前を返す
INDEX_COL (*table-name*, *index-id*, *key_#* [, *userid*] **)**	インデックスカラムの名前を返す
OBJECT_ID (*object-name* **)**	オブジェクト ID を返す
OBJECT_NAME (*object-id* [, *database-id*] **)**	オブジェクトの名前を返す

テキスト関数とイメージ関数

テキスト関数とイメージ関数は、text データ型と image データ型に対して作用します。SQL Anywhere がサポートするテキストとイメージ関数は、textptr のみです。

関数のリスト

次のテキストとイメージ関数を使用できます。

- 「TEXTPTR 関数 [テキストとイメージ]」

関数

関数を 1 つずつリストし、その右側に関数のタイプ (数値、文字など) を示します。

特定のタイプのすべての関数へのリンクについては、「関数のタイプ」139 ページを参照してください。

ABS 関数 [数値]

数値式の絶対値を返します。

構文

ABS(*numeric-expression* **)**

パラメーター

- ***numeric-expression*** 絶対値が返される数値。

戻り値

数値式の絶対値。

数値式データ型	戻り値
INT	INT
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMERIC	NUMERIC

標準と互換性

- **SQL/2008**　ABS 関数は、オプションの SQL/2008 言語機能 T441 の一部です。

例

次の文は、値 66 を返します。

```
SELECT ABS( -66 );
```

ACOS 関数 [数値]

数値式のアークコサインをラジアン単位で返します。

構文

ACOS(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　角度のコサイン。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

参照

- 「ASIN 関数 [数値]」154 ページ
- 「ATAN 関数 [数値]」155 ページ
- 「ATAN2 関数 [数値]」156 ページ
- 「COS 関数 [数値]」182 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、0.52 のアークコサイン値を返します。

```
SELECT ACOS( 0.52 );
```

ARGN 関数 [その他]

引数リストから選択された引数を返します。

構文

ARGN(*integer-expression*, *expression* [, ...] **)**

パラメーター

- ***integer-expression*** 式リスト内での引数の位置。
- ***expression*** 関数に渡される任意のデータ型の式。すべて同じデータ型の式を指定してください。

戻り値

integer-expression の値に n を使用すると、残りの引数リストから n 番目の引数 (1 から開始) が返されます。

備考

式のデータ型は任意ですが、すべて同じデータ型にしてください。integer-expression は、1 からリスト内の式の数までの範囲内で指定してください。範囲外の値を指定すると、NULL が返されます。複数の式は、カンマで区切って指定します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 6 を返します。

```
SELECT ARGN( 6, 1,2,3,4,5,6 );
```

ASCII 関数 [文字列]

文字列式の最初のバイトの ASCII 値を整数で返します。

構文

ASCII(*string-expression* **)**

パラメーター

- ***string-expression*** 文字列。

戻り値

SMALLINT

備考

文字列が空の場合は、0 を返します。リテラル文字列は、引用符で囲んで指定します。データベース文字セットがマルチバイトであり、パラメーター文字列の最初の文字が複数バイトから構成される場合、結果は NULL です。

参照

- 「CHAR 関数 [文字列]」167 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 90 を返します。

```
SELECT ASCII( 'Z' );
```

ASIN 関数 [数値]

数値のアークサインをラジアン単位で返します。

構文

ASIN(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　角度のサイン。

戻り値

DOUBLE

備考

SIN 関数と ASIN 関数は逆変換の演算です。

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

参照

- 「ACOS 関数 [数値]」152 ページ
- 「ATAN 関数 [数値]」155 ページ
- 「ATAN2 関数 [数値]」156 ページ
- 「SIN 関数 [数値]」345 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、0.52 のアークサイン値を返します。

```
SELECT ASIN( 0.52 );
```

ATAN 関数 [数値]

数値のアークタンジェントをラジアン単位で返します。

構文

ATAN(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　角度のタンジェント。

備考

ATAN 関数と TAN 関数は逆変換の演算です。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

参照

- 「ACOS 関数 [数値]」152 ページ
- 「ASIN 関数 [数値]」154 ページ
- 「ATAN2 関数 [数値]」156 ページ
- 「TAN 関数 [数値]」367 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、0.52 のアークタンジェント値を返します。

```
SELECT ATAN( 0.52 );
```

ATAN2 関数 [数値]

2 つの数の比率のアークタンジェントをラジアン単位で返します。

構文

{ **ATN2** | **ATAN2** }(*numeric-expression-1*, *numeric-expression-2*)

パラメーター

- ***numeric-expression-1*** アークタンジェントが計算される比率の分子。
- ***numeric-expression-2*** アークタンジェントが計算される比率の分母。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

参照

- 「ACOS 関数 [数値]」152 ページ
- 「ASIN 関数 [数値]」154 ページ
- 「ATAN 関数 [数値]」155 ページ
- 「TAN 関数 [数値]」367 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、比率 0.52 ～ 0.60 のアークタンジェント値を返します。

```
SELECT ATAN2( 0.52, 0.60 );
```

AVG 関数 [集合]

対象となるローセットの、数値式の平均値またはユニークな値からなるセットの平均値を計算します。

構文 1

AVG([**ALL** | **DISTINCT**] *numeric-expression* **)**

構文 2

AVG([**ALL**] *numeric-expression* **) OVER (** *window-spec* **)**

window-spec :以下の「備考」の構文 2 の説明を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **[ALL] *numeric-expression*** 各グループのローで平均値が計算される式。
- **DISTINCT 句** グループごとにユニークな数値の平均を計算します。

戻り値

グループにローが含まれていない場合は、NULL 値を返します。

引数が DOUBLE の場合は DOUBLE を、それ以外の場合は NUMERIC を返します。

備考

この平均値には、*numeric-expression* が NULL 値のローは含まれません。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。「WINDOW 句」989 ページの *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

この関数ではオーバーフローエラーが発生することがあり、これにより、エラーが返される場合があります。オーバーフローエラーを回避するために、*numeric-expression* で CAST 関数を使用できます。

参照

- 「CAST 関数 [データ型変換]」165 ページ
- 「SUM 関数 [集合]」363 ページ
- 「COUNT 関数 [集合]」183 ページ

標準と互換性

- **SQL/2008** 構文 1 は SQL/2008 標準のコア機能であり、構文 2 はオプションの SQL/2008 言語機能 T611、「Elementary OLAP operations」の一部です。カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、AVG 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。

参照

- 「集合関数と外部参照」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、値 49988.623200 を返します。

```
SELECT AVG( Salary ) FROM Employees;
```

次の文は、Products テーブルの製品価格の平均を返します。

```
SELECT AVG( DISTINCT UnitPrice ) FROM Products;
```

次の文は、サブクエリからの限定式と外部 SELECT ブロックからの外部参照 (p.Quantity) の両方が AVG の引数に含まれているため、SQLSTATE 42W68 エラーが返されます。

```
select * from Products as p
where p.Quantity > ( select avg( 0.5 * p.Quantity + 0.5 * s.Quantity )
                     from SalesOrderItems as s
                     where s.ProductID = p.ProductID )
```

BASE64_DECODE 関数 [文字列]

MIME base64 フォーマットを使用してデータを復号化し、文字列を LONG VARCHAR として返します。

構文

BASE64_DECODE(*string-expression* **)**

パラメーター

- ***string-expression*** 復号化される文字列。文字列は base64 エンコードである必要があることに注意してください。

戻り値

LONG VARCHAR

参照

- 「BASE64_ENCODE 関数 [文字列]」 159 ページ
- 「文字列関数」 148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、Embedded SQL プログラムからイメージテーブルにイメージを挿入します。入力データ (ホスト変数) は base64 エンコードである必要があります。

```
EXEC SQL INSERT INTO images ( image_data ) VALUES ( BASE64_DECODE ( :img ) );
```

BASE64_ENCODE 関数 [文字列]

MIME base64 フォーマットを使用してデータをエンコードし、そのデータを 7 ビット ASCII 文字列として返します。

構文

BASE64_ENCODE(*string-expression* **)**

パラメーター

- ***string-expression*** エンコードされる文字列。

戻り値

LONG VARCHAR

参照

- 「BASE64_DECODE 関数 [文字列]」158 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、イメージを含むテーブルからデータを取り出し、ASCII フォーマットで返します。結果として返される文字列は電子メールメッセージに埋め込まれ、元のイメージを取り出すために受信者によって復号化されます。

```
SELECT BASE64_ENCODE( image_data ) FROM IMAGES;
```

BIT_AND 関数 [集合]

ローグループごとに、指定された式のビット処理 AND を返します。

構文

BIT_AND(*bit-expression* **)**

パラメーター

- ***bit-expression*** 集約されるオブジェクト。この式は、VARBIT 配列、BINARY 値、または INTEGER (BIT や TINYINT などのすべての整数の変形を含む) にできます。

戻り値

引数と同じデータ型。比較される各ビット位置について、すべてのローでそのビット位置に 1 がある場合は 1 を返し、それ以外の場合は 0 を返します。

参照

- 「BIT_OR 関数 [集合]」161 ページ
- 「BIT_XOR 関数 [集合]」163 ページ
- 「ビット処理演算子」11 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、CHAR カラムを含む 4 つのローを生成し、値を VARBIT に変換します。

```
SELECT BIT_AND( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011')
```

結果 0000 は次のように決定されます。

1. ロー 1 (0001) とロー 2 (0111) の間でビット処理 AND が実行され、結果は 0001 になります (両方の値の 4 番目のビットに 1 があります)。
2. 前の比較結果 (0001) とロー 3 (0100) の間でビット処理 AND が実行され、結果は 0000 になります (これらの値の同じビット位置に 1 がありませんでした)。
3. 前の比較結果 (0000) とロー 4 (0011) の間でビット処理 AND が実行され、結果は 0000 になります (これらの値の同じビット位置に 1 がありませんでした)。

BIT_LENGTH 関数 [ビット配列]

配列に格納されているビット数を返します。

構文

BIT_LENGTH(*bit-expression* **)**

パラメーター

- ***bit-expression***　長さを決定するビット式。

戻り値

INT

参照

- 「CHAR_LENGTH 関数 [文字列]」168 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **SQL/1999**　BIT_LENGTH 関数は、SQL/1999 標準のコア機能でした。BIT VARYING データ型は、SQL/1999 標準のオプションの言語機能 F511 でした。SQL/2003 標準では、BIT_LENGTH および BIT VARYING データ型のサポートが削除されました。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、値 8 を返します。

```
SELECT BIT_LENGTH( '01101011' );
```

BIT_OR 関数 [集合]

ローグループごとに、指定された式のビット処理 OR を返します。

構文

BIT_OR(*bit-expression* **)**

パラメーター

- ***bit-expression*** 集約されるオブジェクト。この式は、VARBIT 配列、BINARY 値、または INTEGER (BIT や TINYINT などのすべての整数の変形を含む) にできます。

戻り値

引数と同じデータ型。この関数では、比較される各ビット位置について、いずれかのローでそのビット位置に 1 がある場合は 1 を返し、それ以外の場合は 0 を返します。

参照

- 「BIT_AND 関数 [集合]」 159 ページ
- 「BIT_XOR 関数 [集合]」 163 ページ
- 「ビット処理演算子」 11 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、CHAR カラムを含む 4 つのローを生成し、値を VARBIT に変換します。

```
SELECT BIT_OR( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011')
```

結果 0111 は次のように決定されます。

1. ロー 1 (0001) とロー 2 (0111) の間でビット処理 OR が実行され、結果は 0111 になります。
2. 前の比較結果 (0111) とロー 3 (0100) の間でビット処理 OR が実行され、結果は 0111 になります。
3. 前の比較結果 (0111) とロー 4 (0011) の間でビット処理 OR が実行され、結果は 0111 になります。

BIT_SUBSTR 関数 [ビット配列]

ビット配列の部分配列を返します。

構文

BIT_SUBSTR(*bit-expression* **[,** *start* **[,** *length* **]])**

パラメーター

- ***bit-expression*** 部分配列を抽出するビット配列。
- ***start*** 返す部分配列の開始位置。負の開始位置は、配列の先頭からではなく、末尾からのビット数を指定します。配列の先頭ビットの位置を 1 とします。
- ***length*** 返す部分配列の長さ。正の length は、開始位置から *length* ビット右側の位置で部分配列が終わることを指定し、負の length は、開始位置から最大 *length* ビット左側のビットを返します。

戻り値

LONG VARBIT

備考

start と *length* には、正または負の値を指定できます。負の数と正の数を適切に組み合わせて使用すると、文字列の先頭または末尾のどちらからでも部分配列を取得できます。*length* に負の値を使用しても、部分配列で返されるビットの順序には影響がありません。

length を指定すると、部分配列は指定した長さに限定されます。*start* が 0 で *length* が負でない場合は、start 値として 1 が使用されます。*start* が 0 で *length* が負の場合は、start 値として -1 が使用されます。

length を指定しない場合、配列の末尾までが選択範囲になります。

BIT_SUBSTR 関数も次と同様ですが、より高速です。

```
CAST( SUBSTR( CAST( bit-expression AS VARCHAR ),
start [, length ] )
AS VARBIT )
```

参照

- 「SUBSTRING 関数 [文字列]」361 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、1101 を返します。

```
SELECT BIT_SUBSTR( '001101', 3 );
```

次の文は、10110 を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT BIT_SUBSTR( '01011011101111011111', 2, 5 );
```

次の文は、11111 を返します。

```
SELECT BIT_SUBSTR( '01011011101111011111', -5, 5 );
```

BIT_XOR 関数 [集合]

ローグループごとに、指定された式のビット処理 XOR を返します。

構文

BIT_XOR(*bit-expression* **)**

パラメーター

- ***bit-expression*** 集約されるオブジェクト。この式は、VARBIT 配列、BINARY 値、または INTEGER (BIT や TINYINT などのすべての整数の変形を含む) にできます。

戻り値

引数と同じデータ型。比較される各ビット位置について、奇数個のローでそのビット位置に 1 がある場合は 1 を返し、それ以外の場合は 0 を返します。

参照

- 「BIT_AND 関数 [集合]」159 ページ
- 「BIT_OR 関数 [集合]」161 ページ
- 「ビット処理演算子」11 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、CHAR カラムを含む 4 つのローを生成し、値を VARBIT に変換します。

```
SELECT BIT_XOR( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011')
```

結果 0001 は次のように決定されます。

1. ロー 1 (0001) とロー 2 (0111) の間でビット処理排他的 OR (XOR) が実行され、結果は 0110 になります。

2. 前の比較結果 (0110) とロー 3 (0100) の間でビット処理 XOR が実行され、結果は 0010 になります。

3. 前の比較結果 (0010) とロー 4 (0011) の間でビット処理 XOR が実行され、結果は 0001 になります。

BYTE_LENGTH 関数 [文字列]

文字列のバイト数を返します。

構文

BYTE_LENGTH(*string-expression* **)**

パラメーター

- ***string-expression*** 長さが計算される文字列。

戻り値

INT

備考

string-expression の後続空白スペースは、返される長さに含まれます。

NULL 文字列の戻り値は NULL です。

マルチバイト文字セットの文字列の場合、BYTE_LENGTH の値は、CHAR_LENGTH で返される文字数と異なることがあります。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「CHAR_LENGTH 関数 [文字列]」 168 ページ
- 「DATALENGTH 関数 [システム]」 192 ページ
- 「LENGTH 関数 [文字列]」 266 ページ
- 「文字列関数」 148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。SQL/2008 標準の OCTET_LENGTH 関数に相当します。

例

次の文は、値 12 を返します。

```
SELECT BYTE_LENGTH( 'Test Message' );
```

BYTE_SUBSTR 関数 [文字列]

文字列の部分文字列を返します。部分文字列は、文字ではなくバイトを使用して計算されます。

構文

BYTE_SUBSTR(*string-expression*, *start* [, *length*] **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***string-expression*** 部分文字列が取得される文字列。
- ***start*** 部分文字列の開始位置を示す整数式。正の整数の場合、文字列の先頭から開始します。先頭文字の位置は 1 です。負の整数の場合、文字列の末尾から開始します。最後の文字の位置は -1 です。
- ***length*** 部分文字列の長さを示す整数式。正の *length* は、取得するバイト数を開始位置から**開始する**ことを指定します。負の *length* は、開始位置から最大 *length* バイト左側のバイトを返します。

戻り値

BINARY、VARCHAR、または NVARCHAR。返される値は *string-expression* の型によって異なります。また、戻り値が LONG になるかどうかは、指定した引数によって決まります。たとえば、長さが 32K 未満の定数を指定した場合、LONG は返されません。

備考

length を指定すると、部分文字列は指定したバイト数に制限されます。*start* と *length* には、正または負の値を指定できます。負の数と正の数を適切に組み合わせて使用すると、文字列の先頭または末尾のどちらからでも部分文字列を取得できます。

start が 0 で length が負でない場合は、1 の *start* 値が使用されます。*start* が 0 で *length* が負の場合は、-1 の start 値が使用されます。

参照

- 「SUBSTRING 関数 [文字列]」361 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 Test を返します。

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

CAST 関数 [データ型変換]

指定されたデータ型に変換した式の値を返します。

CAST、CONVERT、HEXTOINT、INTTOHEX 関数を使用すると、16 進値変換を行うことができます。これらの関数の使用の詳細については、「16 進値との変換」6 ページを参照してください。

構文

CAST(*expression* **AS** *datatype* **)**

パラメーター

- ***expression*** 変換される式。
- ***datatype*** ターゲットのデータ型。

戻り値

要求されたデータ型によって異なります。

備考

文字列型の長さを指定しない場合は、データベースサーバーによって適切な長さが選択されます。DECIMAL 変換で精度も位取りも指定しない場合は、データベースサーバーによって適切な値が選択されます。

CAST 関数を使用して文字列をトランケートする場合、string_rtruncation データベースオプションは OFF に設定する必要があります。それ以外の場合は、エラーになります。文字列のトランケートには LEFT 関数を使用することをおすすめします。

参照

- 「データ型変換」127 ページ
- 「CONVERT 関数 [データ型変換]」179 ページ
- 「LEFT 関数 [文字列]」265 ページ

標準と互換性

- **SQL/2008** CAST 関数は、SQL/2008 標準のコア機能です。ただし、SQL Anywhere の CAST では、SQL 標準で許可されていない数多くのデータ型変換がサポートされています。たとえば、SQL Anywhere では整数値を DATE データ型にキャストできますが、SQL 標準ではこのような変換はできません。詳細については、「データ型変換」127 ページを参照してください。

例

次の関数は、文字列を日付として使用することを保証します。

```
SELECT CAST( '2000-10-31' AS DATE );
```

式 1 + 2 の値を計算し、その結果を 1 文字の文字列にキャストします。

```
SELECT CAST( 1 + 2 AS CHAR );
```

CEILING 関数 [数値]

指定した値以上になる最初の整数を返します。正の数値の場合、この処理は丸めとも呼ばれています。

構文

{ **CEILING** | **CEIL** } (*numeric-expression*)

パラメーター

- ***numeric-expression*** 切り上げ値を計算する対象の数値。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

参照

- 「FLOOR 関数 [数値]」232 ページ

標準と互換性

- **SQL/2008** CEILING 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の文は、値 60 を返します。

```
SELECT CEILING( 59.84567 );
```

CHAR 関数 [文字列]

数値の ASCII 値を持つ文字を返します。

構文

CHAR(*integer-expression* **)**

パラメーター

- ***integer-expression*** ASCII 文字に変換される数値。0 ～ 255 の範囲内の数を指定します。

戻り値

VARCHAR

備考

返される文字は、バイナリソート順に従って、現在のデータベース側文字セットの指定した数値式に対応しています。

整数式の値が 255 より大きいか、0 より小さい場合、CHAR は NULL を返します。

参照

- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 Y を返します。

```
SELECT CHAR( 89 );
```

CHAR_LENGTH 関数 [文字列]

文字列の文字数を返します。

構文

CHAR_LENGTH (*string-expression* **)**

パラメーター

- ***string-expression***　長さが計算される文字列。

戻り値

INT

備考

後続空白スペースは、返される長さに含まれます。

NULL 文字列の戻り値は NULL です。

マルチバイト文字セットの文字列の場合、CHAR_LENGTH 関数で返される値は、BYTE_LENGTH 関数で返されるバイト数と異なることがあります。

> **注意**
> データ型が CHAR、VARCHAR、LONG VARCHAR、NCHAR の場合、CHAR_LENGTH 関数と LENGTH 関数を使用すると同じ結果が得られます。ただし、BINARY データ型とビット配列データ型には LENGTH 関数を使用します。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「BYTE_LENGTH 関数 [文字列]」164 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　CHAR_LENGTH は、SQL/2008 標準のコア機能です。 NCHAR データ型の式での CHAR_LENGTH の使用は、オプションの SQL/2008 言語機能 F421 の一部です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、値 8 を返します。

```
SELECT CHAR_LENGTH( 'Chemical' );
```

CHARINDEX 関数 [文字列]

ある文字列内で 1 つの文字列の位置を返します。

構文

CHARINDEX(*string-expression-1*, *string-expression-2* **)**

パラメーター

- ***string-expression-1*** 検索する文字列。
- ***string-expression-2*** 検索される文字列。

戻り値

INT

備考

string-expression-1 の先頭文字の位置を 1 とします。検索される文字列内に、検索する文字列のインスタンスが複数存在する場合、CHARINDEX 関数は最初のインスタンスの位置を返します。

検索される文字列内に、検索する文字列が存在しない場合、CHARINDEX 関数は 0 を返します。

いずれかの引数が NULL の場合、結果も NULL になります。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「SUBSTRING 関数 [文字列]」361 ページ
- 「REPLACE 関数 [文字列]」330 ページ
- 「LOCATE 関数 [文字列]」271 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、姓に K という文字が含まれる場合にのみ、Employees テーブルの Surname カラムと GivenName カラムから姓名を返します。

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX( 'K', Surname ) = 1;
```

返された結果：

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

COALESCE 関数 [その他]

リストの中から NULL でない最初の式を返します。この関数は ISNULL 関数と同じです。

構文

COALESCE(*expression*, *expression* [, ...] **)**

パラメーター

- ***expression*** 任意の式。

  2 つ以上の式を関数に渡します。すべての式は比較可能であることが必要です。

戻り値

この関数の戻り値は、指定した式によって異なります。具体的には、データベースサーバーが関数を評価するとき、まず、式の比較が可能なデータ型を検索します。該当するデータ型が見つかると、データベースサーバーは式を比較し、比較に使用したデータ型で結果を返します。データベースサーバーは、一般に比較が可能なデータ型を見つけることができないと、エラーを返します。

備考

結果が NULL になるのはすべての引数が NULL の場合のみです。

このパラメーターにはスカラ型を指定できますが、同じ型を指定する必要はありません。

データベースサーバープロセスでのこの関数の処理方法の詳細については、「ISNULL 関数 [その他]」260 ページを参照してください。

参照

- 「ISNULL 関数 [その他]」260 ページ

標準と互換性

- **SQL/2008** コア機能。

例

次の文は、値 34 を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT COALESCE( NULL, 34, 13, 0 );
```

COMPARE 関数 [文字列]

代替照合規則に基づいて 2 つの文字列を比較できます。

構文

COMPARE(
string-expression-1,
string-expression-2
[, { *collation-id*
| *collation-name*[(*collation-tailoring-string*)] }]
)

パラメーター

- ***string-expression-1*** 最初の文字列式。
- ***string-expression-2*** 2 番目の文字列式。

  文字列式には、データベース側文字セットでエンコードされた文字のみを指定できます。

- ***collation-id*** 使用するソート順を指定する変数または定数 (整数)。*collation-id* は、組み込みの照合にのみ使用できます。 「SORTKEY 関数 [文字列]」347 ページを参照してください。

  照合名または照合 ID を指定しない場合のデフォルトは、デフォルト Unicode マルチ言語です。

- ***collation-name*** 使用する照合順の名前を指定する文字列または文字変数。また、char_collation または db_collation (たとえば、COMPARE('abc', 'ABC', 'char_collation');) を指定して、データベースが使用している CHAR の照合順を使用できます。同様に、nchar_collation を指定して、データベースで使用している NCHAR の照合順を使用できます。有効な照合名のリストについては、「SORTKEY 関数 [文字列]」347 ページを参照してください。

- ***collation-tailoring-string*** オプションで、文字列の比較を詳細に制御することを目的に、照合の適合化オプション (*collation-tailoring-string*) を指定できます。これらのオプションは、キーワード=値 の形式で、カッコで囲んで指定して、その後ろに照合名を記述します。たとえば、'UCA(locale=es;case=LowerFirst;accent=respect)' のように記述します。これらのオプションの組み合わせを指定する構文は、CREATE DATABASE 文の COLLATION 句を定義する構文と同じです。

> **注意**
> UCA 照合を指定すると、照合の適合化のすべてのオプションがサポートされます。その他の照合の場合、大文字小文字の区別の適合化オプションのみがサポートされます。

参照

- 「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』

戻り値

選択した照合規則に基づく INTEGER。

値	意味
1	*string-expression-1* が *string-expression-2* より大きい
0	*string-expression-1* が *string-expression-2* に等しい
-1	*string-expression-1* が *string-expression-2* より小さい

備考

COMPARE 関数は、データベースのブランク埋め込みが有効になっている場合でも、空の文字列とスペースだけの文字列を同等と見なしません。COMPARE 関数は SORTKEY 関数を使用して、比較に使用する照合キーを生成します。したがって、空の文字列、1 つのスペースを持った文字列、2 つのスペースを持った文字列は、等しいと見なされません。

string-expression-1 または *string-expression-2* が NULL の場合、結果は NULL です。

参照

- 「SORTKEY 関数 [文字列]」347 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例では、COMPARE 関数を使用して 3 つの比較を行います。

```
SELECT COMPARE( 'abc','ABC','UCA(case=LowerFirst)' ),
       COMPARE( 'abc','ABC','UCA(case=Ignore)' ),
       COMPARE( 'abc','ABC','UCA(case=UpperFirst)' );
```

戻り値は -1、0、1 で、それぞれ比較の結果を示します。最初の比較の結果は -1 です。これは、*string-expression-2* ('ABC') が *string-expression-1* ('abc') よりも小さいことを示します。最初の COMPARE 文で、大文字と小文字の区別が LowerFirst に設定されているため、このような結果となります。

COMPRESS 関数 [文字列]

文字列を圧縮し、LONG BINARY 型の値を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

COMPRESS(*string-expression* **[, '***compression-algorithm-alias***'])**

パラメーター

- ***string-expression*** 圧縮される文字列。この関数にはバイナリ値を渡すことができます。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。
- ***compression-algorithm-alias*** 圧縮に使用するアルゴリズムのエイリアス。サポートされている値は zip と gzip です (いずれも同じアルゴリズムに基づいていますが、ヘッダーと後書きが異なります)。

  zip は幅広くサポートされている圧縮アルゴリズムです。gzip は UNIX の gzip ユーティリティと互換性がありますが、zip アルゴリズムには互換性がありません。

  解凍は同じアルゴリズムで実行します。

戻り値

LONG BINARY

備考

COMPRESS によって返される値は判読できません。返された値が元の文字列より長い場合、その最大サイズは元の文字列 + 12 バイトよりも 0.1% を超えて大きくなることはありません。DECOMPRESS 関数を使用して、圧縮された *string-expression* を解凍できます。

圧縮された値をテーブルに格納する場合は、文字セット変換がデータに対して実行されないように、カラムを BINARY または LONG BINARY にしてください。

参照

- 「DECOMPRESS 関数 [文字列]」209 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例では、文字列 'Hello World' を gzip アルゴリズムを使用して圧縮して作成したバイナリ文字列の長さを返します。この例は、圧縮すると値の長さが短くなるかどうかを判断する場合に便利です。

```
SELECT LENGTH( COMPRESS( 'Hello world', 'gzip' ) );
```

CONFLICT 関数 [その他]

カラムが、SQL Remote 環境で統合データベースに対して実行される UPDATE の競合の原因であるかどうかを示します。

構文

CONFLICT(*column-name* **)**

パラメーター

- ***column-name*** 競合をテストされるカラムの名前。

戻り値

カラムが SQL Remote Message Agent によって実行される UPDATE 文の VERIFY リストにあり、またその文の VALUES リストで提供されている値が更新されるローのカラムの元の値と一致しない場合に TRUE を返します。それ以外の場合は、FALSE を返します。

参照

- 「CREATE TRIGGER 文」670 ページ
- 「更新の競合に対するデフォルトの解決」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

CONFLICT 関数は、エラーメッセージを回避するために、SQL Remote RESOLVE UPDATE トリガーで使用します。CONFLICT 関数の使用法を示すために、次のテーブルについて考えます。

```
CREATE TABLE Admin (
 PKey bigint NOT NULL DEFAULT GLOBAL AUTOINCREMENT,
 TextCol CHAR(20) NULL, PRIMARY KEY ( PKey ) );
```

統合データベースとリモートデータベースの両方で、Admin テーブルに次のローがあると想定します。

```
1, 'Initial'
```

ここで、統合データベースでローを次のように更新します。

```
UPDATE Admin SET TextCol = 'Consolidated Update' WHERE PKey = 1;
```

リモートデータベースで、ローを次のように別の値に更新します。

```
UPDATE Admin SET TextCol = 'Remote Update' WHERE PKey = 1;
```

次に、リモートデータベースで dbremote を実行します。これによって、統合データベースで実行される次の文を含むメッセージファイルが生成されます。

```
UPDATE Admin SET TextCol='Remote Update'
VERIFY ( TextCol )
VALUES ( 'Initial' )
WHERE PKey=1;
```

SQL Remote Message Agent を統合データベースで実行し、この UPDATE 文を適用すると、SQL Anywhere は、VERIFY 句と VALUES 句を使用して、RESOLVE UPDATE トリガーが起動するかどうかを決定します。RESOLVE UPDATE トリガーは、統合データベースに対して SQL Remote Message Agent から更新が実行された場合にのみ起動します。次に、RESOLVE UPDATE トリガーを示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
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;
```

統合データベースの TextCol カラムの現在の値 ('Consolidated Update') は、関連付けられているカラムの VALUES 句の値 ('Initial') と一致しないため、RESOLVE UPDATE トリガーが起動します。

PKey カラムはリモートで実行された UPDATE 文で修正されず、このトリガーからアクセスできる OldRemote.PKey 値がないため、このトリガーは失敗します。

CONFLICT 関数は、次の値を返すことによってこのエラーの回避に役立ちます。

●OldRemote.PKey 値がない場合は、FALSE を返します。

●OldRemote.PKey 値があっても OldConsolidated.PKey と一致する場合は、FALSE を返します。

●OldRemote.PKey 値があり、OldConsolidated.PKey と異なる場合は、TRUE を返します。

CONFLICT 関数を使用して、トリガーを次のように書き直し、エラーを回避できます。

```
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;
```

CONNECTION_EXTENDED_PROPERTY 関数 [文字列]

指定したプロパティの値を返します。オプションでプロパティ固有の文字列パラメーターを指定できます。

構文

CONNECTION_EXTENDED_PROPERTY(
{ *property-id* | *property-name* }
[, *property-specific-argument* [, *connection-id*]]
)

パラメーター

- ***property-id*** 接続プロパティの ID。

- ***property-name*** 接続プロパティの名前。サポートされているプロパティ名は次のとおりです。

  - **CharSet** 指定した規格によって決まる接続の CHAR 文字セットのラベルを返します。可能な値には、ASE、IANA、MIME、JAVA、WINDOWS、UTR22、IBM、ICU があります。デフォルトは IANA です。ただし、データベース接続が TDS で作成された場合のデフォルトは ASE です。

  - **NcharCharSet** 指定した規格によって決まる接続の NCHAR 文字セットのラベルを返します。可能な値は CharSet で前述した一覧と同じです。

  - **Progress** 文が実行されている時間に関する情報を返します。*property-specific-argument* を指定し、続けて *connection-id* を指定すると、文の進行状況に固有の情報が返されます。

    次の文とプロシージャーでは、Progress プロパティがサポートされています。

    - BACKUP DATABASE (イメージとアーカイブの両方)
    - LOAD TABLE (USING FILE と USING CLIENT FILE のみ)
    - MESSAGE
    - REORGANIZE TABLE
    - RESTORE DATABASE
    - UNLOAD (すべてのタイプ)
    - sa_table_page_usage システムプロシージャー

- ***property-specific-argument*** 次の接続プロパティと関連付けられたオプションのプロパティ固有の文字列パラメーター。

  - **Progress**

    - **PercentComplete** 文の処理が完了した割合を取得するには、PercentComplete を指定します。

    - **Completed** 完了したユニット数を取得するには、Completed を指定します。

    - **Total** 処理予定のユニットの合計数を取得するには、Total を指定します。

    - **Units** 処理予定のページ数、ロー数、またはバイト数。

    - **Elapsed** 現在の経過時間 (ミリ秒で表示)。

    - **Remaining** 推定残り時間 (ミリ秒) を取得するには、Remaining を指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **Raw**　上記の値すべての組み合わせを上に挙げた順にセミコロンで区切って取得するには、Raw を指定します。例：43;9728;22230;pages;5025;6138。
- **Formatted**　ユーザーが読むことができる形式で表示するには、を参照してください。次に例を示します。

  43% (9728 of 22230 pages) complete after 00:00:05; estimated 00:00:06 remaining

  残り時間がまだ推定されていないか、完了したユニット数が元の推定よりも大きい場合、Remaining 値は空になります。

Formatted を除くすべてのプロパティ固有の引数では、大きいバイト値がキロバイトやメガバイトに変換されません。

- ***connection-id***　データベース接続の接続 ID 番号。この値が指定されていない場合は、現在の接続の ID 番号が使用されます。

戻り値

拡張接続プロパティを返します。戻り値は VARCHAR です。

備考

CONNECTION_EXTENDED_PROPERTY 関数は CONNECTION_PROPERTY に似ています。異なる点は、CONNECTION_EXTENDED_PROPERTY 関数がプロパティ固有の文字列パラメーターをオプションで指定できることです。プロパティ固有の引数の解釈は、最初の引数で指定されたプロパティ ID またはプロパティ名によって異なります。

CONNECTION_EXTENDED_PROPERTY 関数を使用して、任意の接続プロパティ用に値を返すことができます。ただし、拡張の情報は、拡張のプロパティにのみ使用できます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

参照

- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「progress_messages オプション」『SQL Anywhere サーバー データベース管理』
- 「CONNECTION_PROPERTY 関数 [システム]」178 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」203 ページ
- 「DB_PROPERTY 関数 [システム]」208 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、Java 規格によって決まる、現在の接続の CHAR 文字セットを返します。

```
SELECT CONNECTION_EXTENDED_PROPERTY( 'charset', 'Java' );
```

CONNECTION_PROPERTY 関数 [システム]

特定の接続プロパティの値を文字列で返します。

構文

CONNECTION_PROPERTY(
{ *integer-expression-1* | *string-expression* }
[, *integer-expression-2*] **)**

パラメーター

- ***integer-expression-1*** 通常、文字列式を最初の引数として指定する方が便利です。integer-expression を指定する場合は、接続プロパティ ID を使用します。この ID は PROPERTY_NUMBER 関数を使用して調べることができます。

- ***string-expression*** 接続プロパティの名前。プロパティ ID またはプロパティ名のどちらかが指定されている必要があります。

  接続プロパティのリストについては、「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』を参照してください。

- ***integer-expression-2*** 現在のデータベース接続の接続 ID。この引数を省略すると、現在の接続が使用されます。

戻り値

VARCHAR、LONG VARCHAR

備考

2 番目の引数を省略すると、現在の接続が使用されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

参照

- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「PROPERTY_NUMBER 関数 [システム]」308 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、管理されている準備文の数を返します。

```
SELECT CONNECTION_PROPERTY( 'PrepStmt' );
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CONVERT 関数 [データ型変換]

指定されたデータ型に変換した式を返します。

CAST、CONVERT、HEXTOINT、INTTOHEX 関数を使用すると、16 進値変換を行うことができます。これらの関数の使用の詳細については、「16 進値との変換」6 ページを参照してください。

構文

CONVERT(*datatype*, *expression* [, *format-style*] **)**

パラメーター

- ***datatype*** 変換後の式のデータ型。
- ***expression*** 変換される式。
- ***format-style*** 出力値に適用されるスタイルコード。このパラメーターを使用するのは、文字列を日付または時刻のデータ型に変換するとき、またはその反対方向に変換するときです。次の表は、サポートされるスタイルコードと、そのスタイルコードで作成される出力形式の表現です。スタイルコードは世紀を出力形式に含めるかどうかによって 2 つのカラムに分けられます (たとえば、06 と 2006 など)。

100 以上の位なし (yy) のスタイルコード	100 以上の位あり (yyyy) のスタイルコード	出力形式
-	0 または 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 または 109	Mmm dd yyyy hh:nn:ss:sssAA
10	110	mm-dd-yy[yy]
11	111	[yy]yy/mm/dd

100 以上の位なし (yy) のスタイルコード	100 以上の位あり (yyyy) のスタイルコード	出力形式
12	112	[yy]yymmdd
-	13 または 113	dd Mmm yyyy hh:nn:ss:sss (24 時間表記、ヨーロッパのデフォルト、ミリ秒、4 桁の年)
-	14 または 114	hh:nn:ss:sss (24 時間表記)
-	20 または 120	yyyy-mm-dd hh:nn:ss (24 時間表記、ODBC 標準、4 桁の年)
-	21 または 121	yyyy-mm-dd hh:nn:ss.sss (24 時間表記、ODBC 標準、ミリ秒、4 桁の年)

戻り値

指定されたデータ型によって異なります。

備考

format-style 引数を指定しない場合は、スタイルコード 0 が使用されます。

CONVERT 関数を使用して、文字列の解析時にあいまいさがない場合、文字列を DATE、TIME、または TIMESTAMP データ型に変換できます。*format-style* が指定されている場合、データベースサーバーはこれを文字列の解析方法のヒントとして使用します。データベースサーバーが文字列を明確に解析できない場合、エラーが返されます。

各出力記号 (Mmm など) が出力するスタイルの詳細については、「date_format オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

参照

- 「CAST 関数 [データ型変換]」 165 ページ
- 「CSCONVERT 関数 [文字列]」 189 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。CONVERT 関数は、SQL/2008 標準で定義されています。ただし、SQL 標準の CONVERT は、入力文字列式を異なる文字セットにコード変換することを目的としており、この機能は SQL Anywhere で CSCONVERT 機能として実装されています。

例

次の文は、フォーマットスタイルの使用方法を示します。

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 104 ) FROM SalesOrders;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

OrderDate
16.03.2000
20.03.2000
23.03.2000
25.03.2000
...

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 7 ) FROM SalesOrders;
```

OrderDate
Mar 16, 00
Mar 20, 00
Mar 23, 00
Mar 25, 00
...

次の文は、整数への変換を示し、値 5 を返します。

```
SELECT CONVERT( integer, 5.2 );
```

CORR 関数 [集合]

数値のペアのセットの相関係数を返します。

構文

CORR(*dependent-expression*, *independent-expression* **)**

パラメーター

- ***dependent-expression*** 独立変数に影響される変数。
- ***independent-expression*** 結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

dependent-expression と *independent-expression* はどちらも数値です。関数は、*dependent-expression* または *independent-expression* が NULL であるペアを除外した後で、(*dependent-expression*, *independent-expression*) のセットに適用されます。次の計算が行われます。

COVAR_POP (*y*, *x*) / **STDDEV_POP** (*y*) * **STDDEV_POP** (*x*)

y は *dependent-expression* を表し、*x* は *independent-expression* を表します。

参照

- 「集合関数」139 ページ
- 「COVAR_POP 関数 [集合]」187 ページ
- 「STDDEV_POP 関数 [集合]」354 ページ

標準と互換性

- **SQL/2008**　CORR 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、年齢が所得レベルに関連付けられているかどうかを検出するために相関を実行します。この関数は、値 0.44022675645996 を返します。

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM Employees;
```

COS 関数 [数値]

引数で与えられた角度のコサインをラジアン単位で返します。

構文

COS(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　角度 (ラジアン)。

戻り値

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメーターが NULL 値の場合、結果は NULL 値になります。

参照

- 「ACOS 関数 [数値]」152 ページ
- 「COT 関数 [数値]」183 ページ
- 「SIN 関数 [数値]」345 ページ
- 「TAN 関数 [数値]」367 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、角度 0.52 ラジアンのコサイン値を返します。

```
SELECT COS( 0.52 );
```

COT 関数 [数値]

引数で与えられた角度のコタンジェントをラジアン単位で返します。

構文

COT(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　角度 (ラジアン)。

戻り値

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメーターが NULL 値の場合、結果は NULL 値になります。

参照

- 「COS 関数 [数値]」182 ページ
- 「SIN 関数 [数値]」345 ページ
- 「TAN 関数 [数値]」367 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、0.52 のコタンジェント値を返します。

```
SELECT COT( 0.52 );
```

COUNT 関数 [集合]

指定されたパラメーターに従って、グループのロー数をカウントします。

構文 1

COUNT([* | [ALL | DISTINCT] *expression* **])**

構文 2

COUNT([* | [ALL]*expression* **]) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***** 各グループの中のロー数を返します。セマンティック上、COUNT(*) と COUNT() は同義です。
- **[ALL] *expression*** *expression* の値が NULL でないロー数をグループごとに返します。
- **DISTINCT *expression*** *expression* の値が NULL でないすべてのローについて、重複しない *expression* 値の数をグループごとに返します。

戻り値

COUNT 関数は、INT データ型の値を返します。

COUNT は、NULL 値を返しません。グループにローがないか、またはグループにおいて *expression* の値がすべて NULL である場合、COUNT は 0 を返します。

備考

COUNT 関数から返される最大値は、2147483647 です。大規模な結果セットをカウントする場合、結果に含まれるローの数がこの数よりも多くなる可能性がある場合、またはオーバーフローの可能性がある場合には、COUNT_BIG 関数を使用します。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。「WINDOW 句」989 ページの *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「COUNT(*)」『SQL Anywhere サーバー SQL の使用法』
- 「AVG 関数 [集合]」156 ページ
- 「SUM 関数 [集合]」363 ページ
- 「COUNT_BIG 関数 [集合]」185 ページ

標準と互換性

- **SQL/2008** コア機能。Window 関数として使用する場合 (構文 2)、COUNT はオプションの SQL/2008 言語機能 T611、「Elementary OLAP operations」の一部です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、COUNT 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。

例については、「AVG 関数 [集合]」156 ページを参照してください。

参照

- 「集合関数と外部参照」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、ユニークな各都市と、その都市で働く従業員数を返します。

```
SELECT City, COUNT( * ) FROM Employees GROUP BY City;
```

COUNT_BIG 関数 [集合]

指定されたパラメーターに従って、グループのロー数をカウントします。

構文 1

COUNT_BIG([* | [ALL | DISTINCT] *expression*])

構文 2

COUNT_BIG([* | [ALL] *expression*]) OVER (*window-spec*)

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***** 各グループの中のロー数を返します。セマンティック上、COUNT_BIG(*) と COUNT_BIG() は同義です。
- **[ALL] *expression*** *expression* の値が NULL でないロー数をグループごとに返します。
- **DISTINCT *expression*** *expression* の値が NULL でないすべてのローについて、重複しない *expression* 値の数をグループごとに返します。

戻り値

COUNT_BIG は、BIGINT データ型の値を返します。

COUNT_BIG は、NULL 値を返しません。グループにローがないか、またはグループにおいて *expression* の値がすべて NULL である場合、COUNT_BIG は 0 を返します。

備考

大規模な結果セットをカウントする場合、結果に含まれるローの数が多くなる可能性がある場合、またはオーバーフローの可能性がある場合には、COUNT_BIG 関数を使用することをおすすめします。それ以外の場合は、COUNT 関数を使用します。この関数の最大値は 2147483647 です。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。「WINDOW 句」989 ページの *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「AVG 関数 [集合]」156 ページ
- 「SUM 関数 [集合]」363 ページ
- 「COUNT 関数 [集合]」183 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、COUNT_BIG 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。

  例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「集合関数と外部参照」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、ユニークな各都市と、その都市で働く従業員数を返します。

```
SELECT City, COUNT_BIG( * ) FROM Employees GROUP BY City;
```

COUNT_SET_BITS 関数 [ビット配列]

配列でビットが 1 (TRUE) に設定された数値のカウントを返します。

構文

COUNT_SET_BITS(*bit-expression* **)**

パラメーター

- ***bit-expression***　設定されたビットを決定するビット配列。

戻り値

UNSIGNED INT

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

bit-expression が NULL の場合、NULL を返します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 4 を返します。

```
SELECT COUNT_SET_BITS( '00110011' );
```

次の文は、値 12 を返します。

```
SELECT COUNT_SET_BITS( '0011001111111111' );
```

COVAR_POP 関数 [集合]

数値のペアのセットの母共分散を返します。

構文 1

COVAR_POP(*dependent-expression*, *independent-expression* **)**

構文 2

COVAR_POP(*dependent-expression*, *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression*** 独立変数に影響される変数。
- ***independent-expression*** 結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

dependent-expression と *independent-expression* はどちらも数値です。関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression*, *independent-expression*) のペアのセットに適用されます。その後、次の計算が行われます。

```
( SUM( y * x ) - SUM( x ) * SUM( y ) / n ) / n
```

y は *dependent-expression* を表し、*x* は *independent-expression* を表します。

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。「WINDOW 句」989 ページの *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「COVAR_SAMP 関数 [集合]」188 ページ
- 「SUM 関数 [集合]」363 ページ

標準と互換性

- **SQL/2008**　COVAR_POP 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、従業員の年齢と給与間の関連の強さを測定します。この関数は、値 73785.84005866687 を返します。

```
SELECT COVAR_POP( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
FROM Employees;
```

COVAR_SAMP 関数 [集合]

数値のペアのセットの標本共分散を返します。

構文 1

COVAR_SAMP(*dependent-expression*, *independent-expression* **)**

構文 2

COVAR_SAMP(*dependent-expression*, *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression***　独立変数に影響される変数。
- ***independent-expression***　結果に影響する変数。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

dependent-expression と *independent-expression* はどちらも数値です。関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression*, *independent-expression*) のペアのセットに適用されます。

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。「WINDOW 句」989 ページの *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「COVAR_POP 関数 [集合]」187 ページ
- 「SUM 関数 [集合]」363 ページ

標準と互換性

- **SQL/2008**　COVAR_SAMP 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、値 74782.9460054052 を返します。

```
SELECT COVAR_SAMP( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM Employees;
```

CSCONVERT 関数 [文字列]

文字列の文字セットを変換します。

構文

CSCONVERT(
string-expression,
target-charset-string [, *source-charset-string* [, *options*]] **)**

パラメーター

- ***string-expression*** 変換される文字列。
- ***target-charset-string*** 変換後の文字セット。*target-charset-string* には、次のいずれかを指定します。または以下を指定します。
  - **os_charset** データベースサーバーをホストしているオペレーティングシステムが使用する文字セットのエイリアス。
  - **char_charset** データベースで使用される CHAR 文字セットのエイリアス。
  - **nchar_charset** データベースで使用される NCHAR 文字セットのエイリアス。
  - **options** 次のいずれかのオプションを指定できます。
    - **バイトオーダーマーク (BOM) の新規サポート** **read_bom=on** または **read_bom=off** を指定して、読み込みバイトオーダーマークをオンまたはオフにします。**write_bom=on** または **write_bom=off** を指定して、書き込みバイトオーダーマークをオンまたはオフにします。デフォルトの動作は **read_bom=on** と **write_bom=off** です。
- ***source-charset-string*** *string-expression* で使用されている文字セット。デフォルトは db_charset (データベース側文字セット) です。*source-charset-string* には、次のいずれかを指定します。または以下を指定します。
  - **os_charset** データベースサーバーをホストしているオペレーティングシステムが使用する文字セットのエイリアス。
  - **char_charset** データベースで使用される CHAR 文字セットのエイリアス。
  - **nchar_charset** データベースで使用される NCHAR 文字セットのエイリアス。

戻り値

LONG BINARY

備考

SQL Anywhere でサポートされている文字セットのリストを表示するには、次のコマンドを実行します。

```
dbinit -le
```

この関数で使用できる文字セットラベルの詳細については、「サポートされている文字セット」『SQL Anywhere サーバー データベース管理』を参照してください。

参照

- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。SQL/2008 標準では、1 つの文字セットから別の文字セットへの文字列データの変換に CONVERT 関数 (SQL Anywhere の CONVERT 関数と混同しないこと) が使用され、この関数の引数は CSCONVERT のものとは異なります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

このフラグメントは、mytext カラムを繁体文字中国語文字セットから簡体文字中国語文字セットに変換します。

```
SELECT CSCONVERT( mytext, 'cp936', 'cp950' )
FROM mytable;
```

このフラグメントは、mytext カラムをデータベース側文字セットから簡体文字中国語文字セットに変換します。

```
SELECT CSCONVERT( mytext, 'cp936' )
FROM mytable;
```

ファイル名がデータベースに格納される場合は、そのデータベース側文字セットで格納されます。サーバーが読み込みまたは書き込みを行うファイルの名前がデータベースに格納されている場合 (外部ストアドプロシージャーなど)、ファイル名をオペレーティングシステムの文字セットに明示的に変換してから、ファイルにアクセスしてください。データベースに格納されているファイル名をクライアントが取得する場合は、クライアントの文字セットに自動的に変換されるため、明示的な変換は必要ありません。

このフラグメントは、ファイル名カラムの値をデータベース側文字セットからオペレーティングシステムの文字セットに変換します。

```
SELECT CSCONVERT( filename, 'os_charset' )
FROM mytable;
```

ファイル名のリストが入っているテーブルがあります。外部ストアドプロシージャーは、このテーブルのファイル名をパラメーターとして取り出し、そのファイルから直接情報を読み込みます。次の文は、文字セットの変換が必要ない場合に正しく動作します。

```
SELECT MYFUNC( filename )
FROM mytable;
```

mytable 句は、ファイル名カラムが含まれているテーブルを示します。ただし、ファイル名をオペレーティングシステムの文字セットに変換する必要がある場合は、次の文を使用します。

```
SELECT MYFUNC( csconvert( filename, 'os_charset' ) )
FROM mytable;
```

CUME_DIST 関数 [ランキング]

ローのグループ内で 1 つの値の相対位置を計算します。

構文

CUME_DIST( ) OVER (*window-spec* **)**

window-spec :次の「備考」の項を参照してください

戻り値

0 ～ 1 の DOUBLE 値

備考

復号ソートキーは、現在 CUME_DIST 関数では使用できません。他の任意の RANK 関数では復号ソートキーを使用できます。

window-spec の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。Window 関数として使用する場合、ORDER BY 句を指定する必要があります。また PARTITION BY 句を指定できますが、ROWS 句または RANGE 句は指定できません。「WINDOW 句」989 ページの *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「DENSE_RANK 関数 [ランキング]」213 ページ
- 「PERCENT_RANK 関数 [ランキング]」300 ページ
- 「RANK 関数 [ランキング]」311 ページ

標準と互換性

- **SQL/2008**　CUME_DIST 関数は、オプションの SQL/2008 言語機能 T612、「Advanced OLAP operations」の一部です。

例

次の例は、カリフォルニアに住む従業員の給与に関する累積分布を示す結果セットを返します。

```
SELECT DepartmentID, Surname, Salary,
CUME_DIST() OVER (PARTITION BY DepartmentID
ORDER BY Salary DESC) "Rank"
FROM Employees
WHERE State IN ('CA');
```

結果セットは次のとおりです。

DepartmentID	Surname	Salary	Rank
200	Savarino	72300.000	0.333333333333333
200	Clark	45000.000	0.666666666666667
200	Overbey	39300.000	1

DATALENGTH 関数 [システム]

式の結果に必要な基本となる記憶領域の長さ (バイト) を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

DATALENGTH(*expression* **)**

パラメーター

- ***expression*** 通常は、カラム名です。*expression* が文字列定数の場合は、引用符で囲みます。

戻り値

UNSIGNED INT

備考

DATALENGTH 関数の戻り値は次のとおりです。

データ型	DATALENGTH
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	データの長さ
BINARY	データの長さ

この関数では NCHAR の入力と出力がサポートされています。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、CompanyName カラムで最も長い文字列の長さ 27 を返します。

```
SELECT MAX( DATALENGTH( CompanyName ) )
FROM Customers;
```

次の文は、文字列 '8sdofinsv8s7a7s7gehe4h' の長さを返します。

```
SELECT DATALENGTH( '8sdofinsv8s7a7s7gehe4h' );
```

DATE 関数 [日付と時刻]

式を日付に変換し、時間、分、秒を削除します。

日付フォーマットの解釈の制御については、「date_order オプション」『SQL Anywhere サーバー データベース管理』 および 「nearest_century オプション」『SQL Anywhere サーバー データベース管理』 を参照してください。

構文

DATE(*expression* **)**

戻り値

DATE

パラメーター

- ***expression*** 日付フォーマットに変換される値。通常、文字列。

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 1999-01-02 を日付として返します。

```
SELECT DATE( '1999-01-02 21:20:53' );
```

次の文は、SYSOBJECT システムビューにリストされるすべてのオブジェクトについて、作成日を返します。

```
SELECT DATE( creation_time ) FROM SYSOBJECT;
```

DATEADD 関数 [日付と時刻]

日付の単位をその引数に追加することによって求められる TIMESTAMP または TIMESTAMP WITH TIME ZONE の値を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

DATEADD(*date-part*, *integer-expression*, *timestamp-expression* **)**

date-part :
year
| **quarter**
| **month**
| **week**
| **day**
| **dayofyear**
| **hour**
| **minute**
| **second**
| **millisecond**
| **microsecond**

パラメーター

- ***date-part*** *integer-expression* で表される日付の単位。

  使用可能な日付の単位の完全なリストについては、「日付の単位の指定」142 ページを参照してください。

- ***integer-expression*** *timestamp-expression* に追加される *date-part* 値の数。*integer-expression* には任意の数値型を指定できますが、値はトランケートされて INTEGER になることに注意してください。

- ***timestamp-expression*** 変更される TIMESTAMP または TIMESTAMP WITH TIME ZONE の値。

戻り値

timestamp-expression が TIMESTAMP WITH TIME ZONE の場合は TIMESTAMP WITH TIME ZONE を返し、それ以外の場合は TIMESTAMP を返します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、TIMESTAMP 値 1995-11-02 00:00:000.000 を返します。

```
SELECT DATEADD( month, 102, '1987/05/02' );
```

次の文は、TIMESTAMP 値 1987-05-02 04:00:00.000 を返します。

```
SELECT DATEADD( hour, 4, '1987/05/02' );
```

次の文は、TIMESTAMP WITH TIME ZONE 値 1987-05-06 11:33:00.000+04:00 を返します。

```
SELECT DATEADD( day, 4, CAST( '1987/05/02 11:33:00.000000+04:00' as TIMESTAMP WITH TIME
ZONE ));
```

DATEDIFF 関数 [日付と時刻]

2 つの日付間の期間を返します。

構文

DATEDIFF(*date-part*, *date-expression-1*, *date-expression-2* **)**

date-part :
year
| **quarter**
| **month**
| **week**
| **day**
| **dayofyear**
| **hour**
| **minute**
| **second**
| **millisecond**
| **microsecond**

パラメーター

- ***date-part*** 期間を測定する日付の単位を指定します。

  上記の日付オブジェクトから 1 つを選択します。日付の単位の完全なリストについては、「日付の単位の指定」142 ページを参照してください。

- ***date-expression-1*** 期間の開始日。この値を *date-expression-2* から引いて、2 つの引数の間に存在する *date-part* の数を返します。

- ***date-expression-2*** 期間の終了日。この値から *date-expression-1* を引いて、2 つの引数の間に存在する *date-part* の数を返します。

戻り値

年、四半期、月、週、日、通し日数を示す INT。時、分、秒、ミリ秒、マイクロ秒を表す BIGINT。

備考

この関数は、指定した 2 つの日付間に存在する日付の単位の数を計算します。結果は、日付の単位の数を表す (date-expression-2 - date-expression-1) と同値の符号付き整数です。

DATEDIFF 関数の結果が日付の単位の整数倍でない場合、結果は丸められずに切り捨てになります。

day を日付の単位として使用する場合、DATEDIFF 関数は指定された 2 つの時刻の間の午前 0 時の回数を返します。このとき、2 番目の日付は計算に含まれますが、最初の日付は含まれません。

month を日付の単位として使用する場合、DATEDIFF 関数は 2 つの日付の間に存在する月の初日の数を返します。このとき、2 番目の日付は計算に含まれますが、最初の日付は含まれません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

week を日付の単位として使用する場合、DATEDIFF 関数は 2 つの日付の間に存在する日曜日の数を返します。このとき、2 番目の日付は計算に含まれますが、最初の日付は含まれません。

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、1 を返します。

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' );
```

次の文は、102 を返します。

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' );
```

次の文は、0 を返します。

```
SELECT DATEDIFF( day, '00:00', '23:59' );
```

次の文は、4 を返します。

```
SELECT DATEDIFF( day,
  '1999/07/19 00:00',
  '1999/07/23 23:59' );
```

次の文は、0 を返します。

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' );
```

次の文は、1 を返します。

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' );
```

DATEFORMAT 関数 [日付と時刻]

日付式を表す文字列を、指定したフォーマットで返します。

構文

DATEFORMAT(*datetime-expression*, *string-expression* **)**

パラメーター

- ***datetime-expression***　変換される日時。
- ***string-expression***　変換後の日付のフォーマット。

  日付フォーマットの説明については、「timestamp_format オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

  この関数は NCHAR の入力または出力をサポートしています。

戻り値

VARCHAR

備考

string-expression には、許容される任意の日付フォーマットを使用できます。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 Jan 01, 1989 を返します。

```
SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );
```

DATENAME 関数 [日付と時刻]

TIMESTAMP または TIMESTAMP WITH TIME ZONE の値の特定部分の名前 (月の名前 "June" など) を文字列で返します。

構文

DATENAME(*date-part*, *timestamp-expression* **)**

パラメーター

- ***date-part*** 名前が返される日付の単位。

  使用可能な日付の単位の完全なリストについては、「日付の単位の指定」142 ページを参照してください。

- ***timestamp-expression*** 日付の単位名が返される TIMESTAMP または TIMESTAMP WITH TIME ZONE の値。意味のある結果を取得するには、必要な *date-part* を *timestamp-expression* に含めてください。

戻り値

VARCHAR

備考

結果が数値 (23 日など) の場合でも、DATENAME 関数は文字列を返します。日付の単位 TZOffset が指定されていると、DATENAME はオフセットを { + | - }hh:nn 形式の文字列で返します。

参照

- 「DATEPART 関数 [日付と時刻]」199 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 May を返します。

```
SELECT DATENAME( month, '1987/05/02' );
```

DATEPART 関数 [日付と時刻]

TIMESTAMP または TIMESTAMP WITH TIME ZONE の値の一部を返します。

構文

DATEPART(*date-part*, *timestamp-expression* **)**

パラメーター

- ***date-part*** 名前が返される日付の単位。

  使用可能な日付の単位の完全なリストについては、「日付の単位の指定」142 ページを参照してください。

- ***timestamp-expression*** 一部分が返される TIMESTAMP または TIMESTAMP WITH TIME ZONE の値。

戻り値

INT

備考

意味のある結果を取得するには、必要な *date-part* 部分を *timestamp-expression* に含めてください。

曜日に対応する数値は、first_day_of_week オプションの設定によって変わります。デフォルトは日曜日 = 7 です。

参照

- 「first_day_of_week オプション」『SQL Anywhere サーバー データベース管理』
- 「SET 文 [T-SQL]」929 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 5 を返します。

```
SELECT DATEPART( month , '1987/05/02' );
```

次の例では、TableStatistics テーブルを作成し、SalesOrders テーブルに格納されている年ごとの注文の総数を挿入します。

```
CREATE TABLE TableStatistics (
  ID INTEGER NOT NULL DEFAULT AUTOINCREMENT,
```

```
    Year INT,
    NumberOrders INT );
INSERT INTO TableStatistics ( Year, NumberOrders )
    SELECT DATEPART( Year, OrderDate ), COUNT(*)
    FROM SalesOrders
    GROUP BY DATEPART( Year, OrderDate );
```

DATETIME 関数 [日付と時刻]

式を TIMESTAMP 値に変換します。

構文

DATETIME(*expression* **)**

パラメーター

- ***expression*** 変換される式。通常は文字列です。

戻り値

TIMESTAMP

備考

数値を変換しようとするとエラーが返ります。

参照

- 「式」12 ページ
- 「CAST 関数 [データ型変換]」165 ページ
- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、値 1998-09-09 12:12:12.000 のタイムスタンプを返します。

```
SELECT DATETIME( '1998-09-09 12:12:12.000' );
```

DAY 関数 [日付と時刻]

引数の日付を 1 ～ 31 の間の整数で返します。

構文

DAY(*date-expression* **)**

パラメーター

- ***date-expression*** DATE データ型の日付。

戻り値

SMALLINT

備考

DAY 関数は、引数の日付に対応する 1 ～ 31 の間の整数を返します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 12 を返します。

```
SELECT DAY( '2001-09-12' );
```

DAYNAME 関数 [日付と時刻]

日付から曜日の名前を返します。

構文

DAYNAME(*date-expression* **)**

パラメーター

- ***date-expression*** 日付。

戻り値

VARCHAR

備考

返される曜日名は日本語で、日曜日、月曜日、火曜日、水曜日、木曜日、金曜日、土曜日です。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値土曜日を返します。

```
SELECT DAYNAME ( '1987/05/02' );
```

DAYS 関数 [日付と時刻]

DAYS 関数は、TIMESTAMP を操作するか、2 つの TIMESTAMP 値の間の日数を返します。詳細については、以下の「備考」の項を参照してください。

構文 1

DAYS(*timestamp-expression* **)**

構文 2

DAYS(*timestamp-expression*, *timestamp-expression* **)**

構文 3

DAYS(*timestamp-expression*, *integer-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP 値。

- ***integer-expression*** *timestamp-expression* に加算される日数。*integer-expression* が負の場合、*timestamp-expression.* から適切な日数が引かれます。整数式を指定する場合は、*timestamp-expression* を TIME、DATE または TIMESTAMP として明示的にキャストしてください。*timestamp-expression* が TIME 値の場合、現在の日付が使用されます。

  データ型のキャストの詳細については、「CAST 関数 [データ型変換]」165 ページを参照してください。

戻り値

構文 1 または構文 2 では INTEGER。

構文 3 では TIMESTAMP。

備考

DAYS 関数の結果は、その引数によって異なります。DAYS 関数では、引数内の時間、分、および秒は無視されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **構文 1**　1 つの *timestamp-expression* を DAYS 関数に渡すと、0000-02-29 から *timestamp-expression* までの日数を INTEGER として返します。

> **注意**
> 0000-02-29 は実際の日付を指すための値ではありません。DAYS 関数で使用されるデフォルトの日付です。

- **構文 2**　2 つの TIMESTAMP 値を DAYS 関数に渡すと、その間の日数を整数で返します。
- **構文 3**　TIMESTAMP 値と整数を DAYS 関数に渡すと、整数値を *timestamp-expression* 引数に加算した TIMESTAMP 結果を返します。

構文 2 の代わりに、DATEDIFF 関数を使用します。構文 3 の代わりに、DATEADD 関数を使用します。

参照

- 「DATEDIFF 関数 [日付と時刻]」196 ページ
- 「DATEADD 関数 [日付と時刻]」194 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、整数 729889 を返します。

```
SELECT DAYS( '1998-07-13 06:07:12' );
```

次の文は、整数値 -366 を返します。このことは、2 番目の DATE 値が最初の日付より 366 日前の日付であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。

```
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' );
```

次の文は、TIMESTAMP 値 1999-07-14 00:00:00.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 );

SELECT DATEADD( day, 366, '1998-07-13' );
```

DB_EXTENDED_PROPERTY 関数 [システム]

指定したプロパティの値を返します。オプションでプロパティ固有の文字列パラメーターを指定できます。

構文

DB_EXTENDED_PROPERTY(
{ *property-id* | *property-name* }
[, *property-specific-argument*
[, *database-id* | *database-name*]]
)

パラメーター

- ***property-id*** 問い合わせるデータベースプロパティ ID。
- ***property-name*** 問い合わせるデータベースプロパティ名。

  データベースプロパティの完全なリストについては、「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』を参照してください。

- ***property-specific-argument*** 次のデータベースプロパティを使用すると、以下に示すように、プロパティに関する特定の情報を返すための追加の引数を指定できます。

  - **CharSet プロパティ** 規格の名称を指定して、指定した規格におけるデフォルトの CHAR 文字セットのラベルを取得します。可能な値には、ASE、IANA、MIME、JAVA、WINDOWS、UTR22、IBM、ICU があります。規格が指定されない場合、IANA がデフォルトとして使用されます。ただし、データベース接続が TDS によって作成された場合、デフォルトは ASE です。

  - **CatalogCollation プロパティ、Collation プロパティ、NcharCollation プロパティ** これらのプロパティを問い合わせるとき、次の値を *property-specific-argument* として指定すると、照合に固有の情報が返されます。

    - **AccentSensitive** AccentSensitive を指定すると、照合でのアクセント記号の区別の設定を取得できます。たとえば、次の文は、NCHAR 照合でのアクセント記号の区別の設定を返します。

      ```
      SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'AccentSensitive');
      ```

      戻り値は、Ignore、Respect、French のいずれかです。詳細については、「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

    - **CaseSensitivity** CaseSensitivity を指定すると、照合での大文字と小文字の区別の設定を取得できます。戻り値は、Ignore、Respect、UpperFirst、LowerFirst のいずれかです。詳細については、「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

    - **PunctuationSensitivity** PunctuationSensitivity を指定すると、照合での句読表記の区別の設定を取得できます。戻り値は、Ignore、Primary、Quaternary のいずれかです。詳細については、「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

    - **Properties** Properties を指定すると、照合に指定されるすべての適合化オプションが含まれる文字列を取得できます。返される文字列に含まれるキーワードと値の説明については、「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **Specification**　Specification を指定すると、照合で使用される完全照合指定が含まれる文字列を取得できます。返される文字列に含まれるキーワードと値の説明については、「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

○ **DriveType プロパティ**　DB 領域の名前またはファイル ID を指定して、ドライブのタイプを取得します。返される値は、CD、FIXED、RAMDISK、REMOTE、REMOVABLE、UNKNOWN のいずれかです。何も指定しないと、システム DB 領域のドライブのタイプが返されます。指定した DB 領域が存在しない場合、このプロパティ関数は NULL を返します。DB 領域名とともに現在接続されてないデータベースの ID を指定した場合にも、この関数は NULL を返します。

○ **File プロパティ**　DB 領域名を指定して、データベースルートファイルのファイル名 (パスも含む) を取得します。何も指定しないと、システム DB 領域の情報が返されます。指定したファイルが存在しない場合、このプロパティ関数は NULL を返します。

○ **FileSize プロパティ**　DB 領域の名前またはファイル ID を指定して、指定したファイルのサイズをページ数で取得します。temporary を指定して、テンポラリ DB 領域のサイズを返すこともできます。または、translog を指定して、ログファイルのサイズを返すこともできます。何も指定しないと、システム DB 領域のサイズが返されます。指定したファイルが存在しない場合、このプロパティ関数は NULL を返します。

○ **FreePages プロパティ**　DB 領域の名前またはファイル ID を指定して、空きページ数を取得します。temporary を指定して、テンポラリ DB 領域の空きページ数を返すこともできます。または、translog を指定して、ログファイルの空きページ数を返すこともできます。何も指定しないと、システム DB 領域の空きページ数が返されます。指定したファイルが存在しない場合、このプロパティ関数は NULL を返します。

○ **IOParallelism プロパティ**　DB 領域名を指定して、その DB 領域がサポートする同時 I/O 操作の推定回数を取得します。DB 領域が指定されない場合、現在のシステムの DB 領域が使用されます。

○ **MirrorServerState プロパティ**　サーバー名を指定して、ミラーサーバーの接続ステータスを判断します。CONNECTED、DISCONNECTED、INCOMING ONLY、OUTGOING ONLY、または NULL を返します。

○ **MirrorState プロパティ**　サーバー名を指定して、ミラーサーバーの同期ステータスを判断します。SYNCHRONIZING、SYNCHRONIZED、または NULL を返します。

○ **NextScheduleTime プロパティ**　イベント名を指定して、次にスケジュールされている実行時刻を取得します。

- ***database-id***　DB_ID 関数が返すデータベース ID 番号。通常、データベース名が使用されます。

- ***database-name***　DB_NAME 関数から返されるデータベース名。

戻り値

VARCHAR

備考

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

DB_EXTENDED_PROPERTY 関数は、*property-specific-argument* 文字列パラメーターをオプションで指定できる点を除いて、DB_PROPERTY 関数と同じです。*property-specific-argument* の解釈は、最初の引数で指定されたプロパティ ID またはプロパティ名によって異なります。

3 番目の引数を省略すると、現在のデータベースが使用されます。

テーブル名やプロシージャー名などのカタログ文字列を比較する場合、データベースサーバーでは CHAR 照合が使用されます。UCA 照合の場合、カタログ照合は CHAR 照合と同じですが、適合化は大文字と小文字およびアクセント記号が区別せず、句読表記はレベル 1 でソートされるようになります。これまでの照合の場合、カタログ照合は CHAR 照合と同じですが、適合化は大文字と小文字を区別しないようになります。カタログ照合で使用される適合化は明示的に指定できませんが、Specification プロパティを問い合わせて、カタログ文字列を比較するためにデータベースサーバーで使用される完全照合指定を取得することができます。Specification プロパティの問い合わせは、CHAR 照合とカタログ照合の違いを利用する必要がある場合に便利です。たとえば、句読表記を区別しない CHAR 照合を使用し、アップグレードスクリプトを実行するとします。このスクリプトでは、my_procedure というプロシージャーを定義し、myprocedure という古いバージョンを削除しようとします。CHAR 照合を使用すると my_procedure は myprocedure と等価であるため、次の文では目的の結果を得られません。

```
CREATE PROCEDURE my_procedure( ) ...;
IF EXISTS ( SELECT * FROM SYS.SYSPROCEDURE WHERE proc_name = 'myprocedure' )
THEN DROP PROCEDURE myprocedure
END IF;
```

その代わり、次の文を実行すると目的の結果を得られます。

```
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;
```

参照

- 「DB_ID 関数 [システム]」207 ページ
- 「DB_NAME 関数 [システム]」208 ページ
- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「CONNECTION_PROPERTY 関数 [システム]」178 ページ
- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」175 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、現在のデータベースの場所を返します。

```
SELECT DB_EXTENDED_PROPERTY( 'File' );
```

次の文は、システムの DB 領域のファイルサイズをページ単位で返します。

```
SELECT DB_EXTENDED_PROPERTY( 'FileSize' );
```

次の文は、トランザクションログのファイルサイズをページ単位で返します。

```
SELECT DB_EXTENDED_PROPERTY( 'FileSize', 'translog' );
```

たとえば、次の文は、NCHAR 照合での大文字と小文字の区別の設定を返します。

```
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation',' CaseSensitivity' );
```

次の文は、データベースの CHAR 照合に指定された適合化オプションを返します。

```
SELECT DB_EXTENDED_PROPERTY ( 'Collation', 'Properties' );
```

次の文は、データベースの NCHAR 照合の完全照合指定を返します。

```
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'Specification' );
```

次の文は、ミラーサーバー Test の接続ステータスを返します。

```
SELECT DB_EXTENDED_PROPERTY( 'MirrorServerState', 'Test' );
```

次の文は、ミラーサーバー Test の同期ステータスを返します。

```
SELECT DB_EXTENDED_PROPERTY( 'MirrorState', 'Test' );
```

DB_ID 関数 [システム]

データベース ID 番号を返します。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

構文

DB_ID([*database-name*] **)**

パラメーター

- ***database-name*** データベース名を含む文字列。*database-name* を指定しない場合は、現在のデータベースの ID 番号が返されます。

戻り値

INT

参照

- 「global_database_id オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

サーバー上の唯一のデータベースとしての SQL Anywhere サンプルデータベースに対して次の文を実行すると、値 0 を返します。

```
SELECT DB_ID( 'demo' );
```

実行中の唯一のデータベースに対して実行すると、次の文は値 0 を返します。

```
SELECT DB_ID( );
```

DB_NAME 関数 [システム]

指定した ID 番号を持つデータベースの名前を返します。

構文

DB_NAME([*database-id*])

パラメーター

- ***database-id*** データベースの ID。*database-id* には、数値式を指定してください。

戻り値

VARCHAR

備考

データベース ID を指定しない場合は、現在のデータベース名が返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

参照

- 「sa_db_list システムプロシージャー」1067 ページ
- 「NEXT_DATABASE 関数 [システム]」289 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

サーバー上の唯一のデータベースとしての SQL Anywhere サンプルデータベースに対して次の文を実行すると、データベース名 demo を返します。

```
SELECT DB_NAME( 0 );
```

DB_PROPERTY 関数 [システム]

指定したプロパティの値を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

DB_PROPERTY(
{ *property-id* | *property-name* }
[, *database-id* | *database-name*]
)

パラメーター

- ***property-id*** データベースプロパティ ID。
- ***property-name*** データベースプロパティ名。
- ***database-id*** DB_ID 関数が返すデータベース ID 番号。通常、データベース名が使用されます。
- ***database-name*** DB_NAME 関数から返されるデータベース名。

戻り値

VARCHAR、LONG VARCHAR

備考

文字列を返します。2 番目の引数を省略すると、現在のデータベースが使用されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

参照

- 「DB_ID 関数 [システム]」207 ページ
- 「DB_NAME 関数 [システム]」208 ページ
- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「PROPERTY 関数 [システム]」304 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、現在のデータベースのページサイズをバイト単位で返します。

```
SELECT DB_PROPERTY( 'PageSize' );
```

DECOMPRESS 関数 [文字列]

文字列を解凍し、LONG BINARY 値を返します。

構文

DECOMPRESS(*string-expression* [, *compression-algorithm-alias*] **)**

パラメーター

- ***string-expression*** 解凍する文字列。この関数にはバイナリ値を渡すこともできます。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

- ***compression-algorithm-alias*** 解凍に使用するアルゴリズムのエイリアス (文字列)。サポートされている値は zip と gzip です (いずれも同じアルゴリズムに基づいていますが、ヘッダーと後書きが異なります)。

  zip は幅広くサポートされている圧縮アルゴリズムです。gzip は UNIX の gzip ユーティリティと互換性がありますが、zip アルゴリズムには互換性がありません。

  アルゴリズムが指定されない場合、文字列の圧縮に使用されたアルゴリズムの検出が試行されます。指定したアルゴリズムが正しくない場合、または正しいアルゴリズムが検出できなかった場合、文字列は解凍されません。

  圧縮の詳細については、「COMPRESS 関数 [文字列]」172 ページを参照してください。

戻り値

LONG BINARY

備考

この関数を使用して、COMPRESS 関数で圧縮された値を解凍できます。

圧縮されているカラムに格納されている値には DECOMPRESS 関数を使用する必要はありません。圧縮されているカラム値の圧縮と解凍は、データベースサーバーが自動的に処理します。

参照

- 「カラムの圧縮の考慮事項」『SQL Anywhere サーバー データベース管理』
- 「COMPRESS 関数 [文字列]」172 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例では、DECOMPRESS 関数を使用して、架空のテーブル TableA の Attachment カラムの値を解凍します。

```
SELECT DECOMPRESS ( Attachment, 'gzip' )
FROM TableA;
```

元の値が LONG VARCHAR などの文字型だった場合、DECOMPRESS はバイナリ値を返すため、CAST を適用して人間が解読できる値を返すようにできます。

```
SELECT CAST ( DECOMPRESS ( Attachment, 'gzip' )
AS LONG VARCHAR ) FROM TableA;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DECRYPT 関数 [文字列]

指定されたキーを使用して文字列を復号化し、LONG BINARY 値を返します。

構文

DECRYPT(*string-expression*, *key*
[, *algorithm*]
)

algorithm :
'AES'
| **'AES256'**
| **'AES_FIPS'**
| **'AES256_FIPS'**

パラメーター

- ***string-expression*** 復号化される文字列。この関数にはバイナリ値を渡すこともできます。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

- ***key*** *string-expression* の復号化に必要な暗号化キー (文字列)。暗号化された元の値を取得するには、このキーが、*string-expression* の暗号化に使用された暗号化キーと同じである必要があります。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

  警告
  強力な暗号化が適用されたデータベースの場合、キーのコピーは必ず安全な場所に保管してください。暗号化キーがわからなくなった場合は、Sybase 製品の保守契約を結んでいるサポートセンタに依頼してもデータにはアクセスできません。アクセスできなくなったデータベースは、廃棄して、新しくデータベースを作成する必要があります。

- ***algorithm*** このオプションのパラメーターでは、*string-expression* の暗号化に使用されたアルゴリズムを指定します。

戻り値

LONG BINARY

備考

サポートされている暗号化アルゴリズムの詳細については、「ENCRYPT 関数 [文字列]」216 ページを参照してください。

DECRYPT 関数を使用して、ENCRYPT 関数で暗号化された *string-expression* を復号化できます。この関数は、入力文字列と同じバイト数の LONG BINARY 値を返します。

string-expression を正常に復号化するには、データの暗号化に使用したのと同じ暗号化キーを使用します。不正な暗号化キーを指定した場合は、エラーが生成されます。キーを紛失すると、データにアクセスできなくなり、そこからのリカバリも不可能になります。

> **注意**
> FIPS 認定の暗号化をサポートしないプラットフォームもあります。サポートされるプラットフォームのリストについては、http://www.ianywhere.jp/tech/1061806-os_components.html を参照してください。

参照

- 「ENCRYPT 関数 [文字列]」216 ページ
- 「ISENCRYPTED 関数 [システム]」259 ページ
- 「データベース部分の暗号化」『SQL Anywhere サーバー データベース管理』
- 「文字列関数」148 ページ
- 「-fips dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、user_info テーブルにあるユーザーのパスワードを復号化します。DECRYPT 関数は値を LONG BINARY データ型に変換して判読不能にするため、パスワードを CHAR データ型に変換するには CAST 関数を使用します。

```
SELECT CAST( DECRYPT( user_pwd, '8U3dkA' ) AS CHAR(100) ) FROM user_info;
```

DEGREES 関数 [数値]

数値をラジアンから度数に変換します。

構文

DEGREES(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　角度 (ラジアン)。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、*numeric-expression* で指定される角度を返します。パラメーターが NULL 値の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 29.79380534680281 を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT DEGREES( 0.52 );
```

DENSE_RANK 関数 [ランキング]

パーティション内の値のランクを計算します。同位の値の場合、DENSE_RANK はランキングシーケンス内にギャップを残しません。

構文

DENSE_RANK( ) OVER (*window-spec* **)**

window-spec :次の「備考」の項を参照してください

戻り値

INTEGER

備考

window-spec の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。Window 関数として使用する場合、ORDER BY 句を指定する必要があります。また PARTITION BY 句を指定できますが、ROWS 句または RANGE 句は指定できません。詳細については、WINDOW 句の *window-spec* 定義を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「CUME_DIST 関数 [ランキング]」191 ページ
- 「PERCENT_RANK 関数 [ランキング]」300 ページ
- 「RANK 関数 [ランキング]」311 ページ
- 「Window 関数」『SQL Anywhere サーバー SQL の使用法』
- 「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　DENSE_RANK 関数は、オプションの SQL/2008 言語機能 T612、「Advanced OLAP operations」の一部です。

  SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」がサポートされています。これにより、カラム参照ではない任意の式を Window 関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、DENSE_RANK 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例は、ユタとニューヨークの従業員の給与ランキングを示す結果セットを返します。結果セットには 19 レコードが返されますが、リスト内の 7 番目と 8 番目の従業員は同一給与で 7 位の同順であるため、18 のランキングのみリストされています。DENSE_RANK 関数はランクにギャップを残さないため、9 番目の従業員を '9' とランキングする代わりに、その従業員は '8' とリストされます。

```
SELECT DepartmentID, Surname, Salary, State,
DENSE_RANK() OVER (ORDER BY Salary DESC) AS SalaryRank
FROM Employees
WHERE State IN ('NY','UT');
```

結果セットは次のとおりです。

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DepartmentID	Surname	Salary	State	SalaryRank
300	Bigelow	31200.000	UT	17
500	Lynch	24903.000	UT	18

DIFFERENCE 関数 [文字列]

2 つの文字列式の SOUNDEX 値の差を返します。

構文

DIFFERENCE (*string-expression-1*, *string-expression-2* **)**

パラメーター

- ***string-expression-1*** 最初の SOUNDEX 引数。
- ***string-expression-2*** 2 番目の SOUNDEX 引数。

戻り値

SMALLINT

備考

DIFFERENCE 関数は 2 つの文字列の SOUNDEX 値を比較し、値の類似性を評価し、0 ～ 4 の値を返します。最も一致する場合は 4 です。

この関数は常に何らかの値を返します。結果が NULL になるのは引数の 1 つが NULL の場合のみです。

参照

- 「SOUNDEX 関数 [文字列]」350 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、test と chest という単語の類似点を返します。

```
SELECT DIFFERENCE( 'test', 'chest' );
```

DOW 関数 [日付と時刻]

指定した日付の曜日を表す 1 ～ 7 の数を返します (日曜日 = 1、月曜日 = 2、以下同様)。

構文

DOW(*date-expression* **)**

パラメーター

- ***date-expression*** 評価される値 (DATE データ型)。

戻り値

SMALLINT

備考

DOW 関数は、first_day_of_week データベースオプションで指定された値の影響を受けません。たとえば、first_day_of_week が月曜日に設定されている場合でも、DOW 関数は月曜日に 2 を返します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 5 を返します。

```
SELECT DOW( '1998-07-09' );
```

次の文は、値 1 を返します。

```
SELECT DOW( CAST( '2010/05/30 11:33:00.000000+04:00' as TIMESTAMP WITH TIME ZONE ));
```

次の文は、Employees テーブルに問い合わせ、週の曜日を表す数として StartDate を返します。

```
SELECT DOW( StartDate ) FROM Employees;
```

ENCRYPT 関数 [文字列]

指定された暗号化キーを使用して指定された値を暗号化し、LONG BINARY 値を返します。

構文

ENCRYPT(*string-expression*, *key*
[, *algorithm*]
)

algorithm :
'AES'
| **'AES256'**
| **'AES_FIPS'**
| **'AES256_FIPS'**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***string-expression*** 暗号化されるデータ。この関数にはバイナリ値を渡すこともできます。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

- ***key*** *string-expression* の暗号化に使用する暗号化キー。元の値を取得するには、同じキーを使用して値を復号化します。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

  ほとんどのパスワードと同様、最善の方法は、簡単には推測できないキー値を選択することです。キーには最低でも 16 文字の値を選択し、大文字と小文字、数字、文字、特殊文字を組み合わせて使用することをおすすめします。このキーは、データを復号化するたびに必要になります。

  警告
  強力な暗号化が適用されたデータベースの場合、キーのコピーは必ず安全な場所に保管してください。暗号化キーがわからなくなった場合は、Sybase 製品の保守契約を結んでいるサポートセンタに依頼してもデータにはアクセスできません。アクセスできなくなったデータベースは、廃棄して、新しくデータベースを作成する必要があります。

- ***algorithm*** このオプションのパラメーターでは、*string-expression* が暗号化されるときに使用されるアルゴリズムを指定します。Rijndael は、強力な暗号化に使用されているアルゴリズムです。これは、米国商務省標準技術局 (NIST：National Institute of Standards and Technology) によってブロック暗号のための新しい次世代標準暗号化方式 (AES：Advanced Encryption Standard) として選択された、ブロック暗号化アルゴリズムです。

  FIPS 認定の暗号化をサポートしているプラットフォームでは、*algorithm* に FIPS 認定のアルゴリズムのいずれかを指定できます。

  注意
  RSA、ECC、FIPS 認定の各暗号化は、すべてのプラットフォームで使用できるわけではありません。各プラットフォームでサポートされている暗号化方法については、http://www.ianywhere.jp/tech/1061806-os_components.html を参照してください。

  algorithm が指定されない場合、デフォルトで AES が使用されます。データベースサーバーの起動時に -fips サーバーオプションが使用された場合は、AES の代わりに AES_FIPS がデフォルトとして使用されます。

戻り値

LONG BINARY

備考

この関数が返す LONG BINARY 値は、*string-expression* の入力値より最大で 31 バイト長くなります。この関数によって返される値は判読できません。DECRYPT 関数を使用して、ENCRYPT 関数で暗号化された *string-expression* を復号化できます。*string-expression* を正常に復号化するには、データの暗号化に使用したのと同じ暗号化キーとアルゴリズムを使用します。不正な暗号化

キーを指定した場合は、エラーが生成されます。キーを紛失すると、データにアクセスできなくなり、そこからのリカバリも不可能になります。

暗号化された値をテーブルに格納する場合は、文字セット変換がデータに対して実行されないように、カラムを BINARY または LONG BINARY にしてください。

参照

- 「DECRYPT 関数 [文字列]」211 ページ
- 「ISENCRYPTED 関数 [システム]」259 ページ
- 「データベース部分の暗号化」『SQL Anywhere サーバー データベース管理』
- 「-fips dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次のトリガーは、user_info テーブルの user_pwd カラムを暗号化します。このカラムにはユーザーのパスワードが含まれ、トリガーは、パスワード値が変更されるたびに起動します。

```
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;
```

ERRORMSG 関数 [その他]

現在のエラー、または指定した SQLSTATE 値または SQLCODE 値のエラーメッセージを返します。

構文

ERRORMSG([*sqlstate* | *sqlcode*] **)**

sqlstate: *string*

sqlcode: *integer*

パラメーター

- ***sqlstate***　この SQLSTATE 値のエラーメッセージが返されます。
- ***sqlcode***　この SQLCODE 値のエラーメッセージが返されます。

戻り値

エラーメッセージを含む VARCHAR。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

引数を指定しない場合は、現在のステータスのエラーメッセージが返されます。テーブル名やカラム名などが代入されます。

引数を指定した場合は、指定した SQLSTATE または SQLCODE のエラーメッセージが返され、代入は行われません。テーブル名とカラム名は、プレースホルダー (%1) で指定されます。

参照

- 「SQL Anywhere のエラーメッセージ (SQLSTATE 順)」『エラーメッセージ』
- 「SQL Anywhere のエラーメッセージ (SQLCODE 順)」『エラーメッセージ』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、SQLCODE -813 のエラーメッセージを返します。

```
SELECT ERRORMSG( -813 );
```

ESTIMATE 関数 [その他]

指定したパラメーターに基づいてクエリオプティマイザーによって計算されたパーセンテージとして、選択性推定を返します。

構文

ESTIMATE(*column-name* [, *value* [, *relation-string*]] **)**

パラメーター

- ***column-name***　推定で使用されるカラム。
- ***value***　カラムが比較される値。デフォルト値は NULL です。
- ***relation-string***　比較に使用される比較演算子。一重引用符で囲んで指定します。このパラメーターに使用できる値は、'='、'>'、'<'、'>='、'<='、'<>'、'!='、'!<'、'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '!>' です。デフォルトは '=' です。

戻り値

REAL

備考

この関数は、述部 column-name relation-string value の選択性推定を返します。*value* が NULL で、比較演算子が '=' の場合は、述部 column-name IS NULL の選択性推定になります。*value* が NULL で、比較演算子が '!=' または '<>' の場合は、述部 column-name IS NOT NULL の選択性推定になります。

参照

- 「選択性推定ソース」『SQL Anywhere サーバー SQL の使用法』
- 「グラフィカルなプラン内の選択性情報」『SQL Anywhere サーバー SQL の使用法』
- 「INDEX_ESTIMATE 関数 [その他]」256 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」220 ページ
- 「EXPERIENCE_ESTIMATE 関数 [その他]」227 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、200 より大きい値を持つことが推定される EmployeeID 値の割合を返します。正確な値は、データベースで実行したアクションに応じて異なります。

```
SELECT FIRST ESTIMATE( EmployeeID, 200, '>' )
  FROM Employees
  ORDER BY 1;
```

ESTIMATE_SOURCE 関数 [その他]

クエリオプティマイザーで使用される選択性推定のソースを提供します。

構文

ESTIMATE_SOURCE(
column-name
[, *value*
[, *relation-string*]]
)

パラメーター

- ***column-name***　調査されるカラムの名前。
- ***value***　カラムが比較される値。デフォルト値は NULL です。
- ***relation-string***　比較に使用される比較演算子。一重引用符で囲んで指定します。このパラメーターに使用できる値は、'='、'>'、'<'、'>='、'<='、'<>'、'!='、'!<'、'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '!>' です。デフォルトは '=' です。

戻り値

次のリストは、ESTIMATE_SOURCE から返される選択性推定のソースを示します。ソースの詳細については、「選択性推定ソース」『SQL Anywhere サーバー SQL の使用法』を参照してください。

値	選択性推定のソース
Statistics	格納されているカラム統計

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

値	選択性推定のソース
Column	カラム統計に格納されているすべての値の平均
Index	インデックス調査
Guess	各タイプの述部に定義された組み込み規則。この値は、使用する関連インデックスが存在しない場合、参照カラムに対して統計が一切収集されていない場合、または述部が複雑である場合にのみ返されます。
Computed	上記以外のソース
Always	指定の述部が常に true の場合に返されます。
Combined	上記の 1 つ以上のソース
Bounded	選択性推定に上限値や下限値が配置されている場合に返されます。

備考

この関数は、述部 column-name relation-string value の選択性推定のソースを返します。*value* が NULL で、比較演算子が '=' である場合、述部 column-name IS NULL に対する選択性推定のソースが返されます。*value* が NULL で、比較演算子が '!=' または '<>' の場合は、述部 column-name IS NOT NULL に対する選択性推定のソースが返されます。

参照

- 「選択性推定ソース」『SQL Anywhere サーバー SQL の使用法』
- 「ESTIMATE 関数 [その他]」219 ページ
- 「INDEX_ESTIMATE 関数 [その他]」256 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、EmployeeID カラムの最初の値が 200 より大きいかどうかを評価するための選択性ソースのインデックスを返します。インデックスが返されることは、クエリオプティマイザーがインデックスを使用して選択性を推定したことを意味します。

```
SELECT FIRST ESTIMATE_SOURCE( EmployeeID, 200, '>' )
  FROM Employees
  ORDER BY 1;
```

EVENT_CONDITION 関数 [システム]

イベントハンドラーがトリガーされる条件を指定します。

構文

EVENT_CONDITION(*condition-name* **)**

パラメーター

- ***condition-name*** イベントをトリガーする条件。指定可能な値はデータベースにあらかじめ設定されています。大文字と小文字は区別されません。各条件は、特定のイベントタイプにのみ有効です。次の表は、各条件とそれらが有効となるイベントを示します。

条件名	単位	イベント	コメント
DBFreePercent	なし	DBDiskSpace	
DBFreeSpace	MB	DBDiskSpace	
DBSize	MB	GrowDB	
ErrorNumber	なし	RAISERROR	
IdleTime	秒	ServerIdle	
Interval	秒	すべて	ハンドラーが最後に実行してから経過した時間
LogFreePercent	なし	LogDiskSpace	
LogFreeSpace	MB	LogDiskSpace	
LogSize	MB	GrowLog	
RemainingValues	integer	GlobalAutoincrement	残りの値の数
TempFreePercent	なし	TempDiskSpace	
TempFreeSpace	MB	TempDiskSpace	
TempSize	MB	GrowTemp	

戻り値

INT

備考

イベントから呼び出されていない場合、EVENT_CONDITION 関数は NULL を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CREATE EVENT 文」540 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次のイベント定義は、EVENT_CONDITION 関数を使用します。

```
CREATE EVENT LogNotifier
TYPE LogDiskSpace
WHERE event_condition( 'LogFreePercent' ) < 50
HANDLER
BEGIN
  MESSAGE 'LogNotifier message'
END;
```

EVENT_CONDITION_NAME 関数 [システム]

EVENT_CONDITION に指定可能なパラメーターをリストします。

構文

EVENT_CONDITION_NAME(*integer* **)**

パラメーター

- ***integer***　0 以上の整数。

戻り値

VARCHAR

備考

関数が NULL を返すまで整数をループすると、EVENT_CONDITION_NAME 関数を使用して EVENT_CONDITION 関数のすべての引数のリストを取得できます。

イベントから呼び出されていない場合、EVENT_CONDITION_NAME 関数は NULL を返します。

参照

- 「CREATE EVENT 文」540 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

EVENT_PARAMETER 関数 [システム]

イベントハンドラーのためのコンテキスト情報を提供します。

構文

EVENT_PARAMETER(*context-name* **)**

context-name :
AppInfo
| **ConnectionID**
| **DisconnectReason**
| **EventName**
| **Executions**
| **MirrorServerName**
| **NumActive**
| **ScheduleName**
| **SQLCODE**
| **TableName**
| **User**
| *condition-name*

パラメーター

- ***context-name*** あらかじめ設定されている文字列の 1 つ。文字列は引用符で囲み、大文字と小文字を区別しません。次の情報を受け渡します。

  - **AppInfo** イベントをトリガーさせた接続の AppInfo 接続プロパティの値。イベントのコンテキスト外のプロパティの値を参照するには、次の文を使用します。

    ```
    SELECT CONNECTION_PROPERTY( 'AppInfo' );
    ```

    AppInfo 文字列には、Embedded SQL、ODBC、OLE DB、ADO.NET、SQL Anywhere JDBC ドライバーの各接続に対するクライアント接続コンピューター名とアプリケーション名が含まれています。

  - **ConnectionID** イベントをトリガーさせた接続の接続 ID。

  - **DisconnectReason** 接続が終了した理由を示す文字列。このパラメーターは、Disconnect イベントに対してのみ有効です。表示される結果は、次のとおりです。

    - **abnormal** データベース接続を切断する前にクライアントアプリケーションが異常終了したか、またはクライアントコンピューターとサーバーコンピューターの間で通信エラーが発生したことによって、切断が行われました。

    - **connect failed** 接続の試行に失敗しました。

    - **drop connection** DROP CONNECTION 文が実行されました。

    - **from client** クライアントアプリケーションが接続を切断しました。

    - **inactive** -ti サーバーオプションで指定された期間に、要求が受信されませんでした。

    - **liveness** -tl サーバーオプションで指定された期間に、活性パケットが受信されませんでした。

  - **EventName** トリガーされたイベント名。

  - **Executions** イベントハンドラーの実行回数。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **MirrorServerName**　データベースミラーリングシステムのプライマリサーバーとの接続を失ったミラーサーバーまたは監視サーバーの名前。

○ **NumActive**　イベントハンドラーのアクティブインタンス数。これは、一定時間に 1 つのイベントハンドラーで 1 つのインスタンスだけを実行させるように制限する場合に利用できます。

○ **ScheduleName**　イベントを起動させたスケジュール名。イベントが、TRIGGER EVENT を使用して手動で起動された場合、またはシステムイベントとして起動された場合、結果は空の文字列になります。スケジュールが作成されたときにスケジュール名が明示的に割り当てられなかった場合は、イベントの名前になります。

○ **SQLCODE**　接続の失敗時に発生したエラーの SQLCODE。このパラメーターは、ConnectFailed イベントに対してのみ有効です。

○ **TableName**　RemainingValues で使用するテーブル名。

○ **User**　イベントをトリガーさせたユーザーのユーザー ID。

さらに、EVENT_PARAMETER 関数からは、EVENT_CONDITION 関数の有効なすべての *condition-name* 引数にアクセスできます。

次の表は、システムイベントタイプに対して有効な context-name 値を示します。

システムイベントタイプ	Context-name 値
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

戻り値

VARCHAR

備考

イベントに渡される値の最大サイズは、サーバーの最大ページサイズ (-gp サーバーオプション) によって制限されます。このサイズより長い値は、最大ページサイズを下回るサイズにトランケートされます。

参照

- 「EVENT_CONDITION 関数 [システム]」222 ページ
- 「CREATE EVENT 文」540 ページ
- 「TRIGGER EVENT 文」960 ページ
- 「-gp dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、イベントに文字列パラメーターを渡す方法を示します。イベントは、トリガーされた時刻をデータベースサーバーメッセージウィンドウに表示します。

```
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' || event_parameter( 'time' );
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string(current timestamp ) );
```

EXP 関数 [数値]

自然対数の底 e を、与えられた引数でべき乗した結果を返します。

構文

EXP(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　指数。

戻り値

DOUBLE

備考

EXP 関数は、自然対数の底 e を *numeric-expression* で指定された値でべき乗した結果を返します。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメーターが NULL 値の場合、結果は NULL 値になります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　EXP 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の文は、値 3269017.3724721107 を返します。

```
SELECT EXP( 15 );
```

EXPERIENCE_ESTIMATE 関数 [その他]

指定したパラメーターに基づいてクエリオプティマイザーによって計算されたパーセンテージとして、選択性推定を返します。

構文

EXPERIENCE_ESTIMATE(
column-name
[, *value*
[, *relation-string*]]
)

パラメーター

- ***column-name***　調査されるカラムの名前。
- ***value***　カラムが比較される値。
- ***relation-string***　比較に使用される比較演算子。このパラメーターに使用できる値は、'='、'>'、'<'、'>='、'<='、'<>'、'!='、'!<'、'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '!>' です。デフォルトは '=' です。

戻り値

REAL

備考

value が NULL の場合、比較演算子 = と != はそれぞれ IS NULL 条件と IS NOT NULL 条件として解釈されます。

参照

- 「ESTIMATE 関数 [その他]」219 ページ
- 「INDEX_ESTIMATE 関数 [その他]」256 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」220 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、NULL を返します。

```
SELECT DISTINCT EXPERIENCE_ESTIMATE( EmployeeID, 200, '>' )
FROM Employees;
```

EXPLANATION 関数 [その他]

SQL 文の最適化方法をプレーンテキスト文字列で返します。

構文

EXPLANATION(
string-expression
[, *cursor-type*]
[, *update-status*]
)

パラメーター

- ***string-expression*** SQL 文。通常は SELECT 文ですが、UPDATE 文、MERGE 文、または DELETE 文も指定できます。
- ***cursor-type*** カーソルタイプ。文字列として表現されます。使用できる値は、asensitive、insensitive、sensitive、または keyset-driven です。*cursor-type* が指定されない場合、デフォルトで asensitive が使用されます。
- ***update-status*** 次のいずれかの値を受け入れる文字列パラメーター。これらの値は、指定されたカーソルをオプティマイザーがどのように処理するかを示します。

値	説明
READ-ONLY	このカーソルは読み込み専用。
READ-WRITE (デフォルト)	このカーソルは読み込みや書き込みが可能。
FOR UPDATE	このカーソルは読み込みや書き込みが可能。READ-WRITE と同じです。

戻り値

LONG VARCHAR

備考

文のアクセスプランが文字列で返されます。オプションの詳細については、「高度：クエリ実行プラン」『SQL Anywhere サーバー SQL の使用法』 を参照してください。

GRAPHICAL_PLAN 関数では、文の最適化方法の影響を受けるシステムプロパティなど、アクセスプランに関するさまざまな情報が提供されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

この情報は、追加するインデックスの決定や、パフォーマンスを向上するためのデータベース構造の決定に役立ちます。

参照

- 「Ultra Light の実行プラン」『Ultra Light データベース管理とリファレンス』
- 「高度：クエリ実行プラン」『SQL Anywhere サーバー SQL の使用法』
- 「PLAN 関数 [その他]」302 ページ
- 「GRAPHICAL_PLAN 関数 [その他]」236 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、SELECT 文を文字列パラメーターとして渡し、クエリを実行するためのプランを返します。

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

次の文は、'select * from Department where ....' クエリに対する INSENSITIVE カーソルの短いプランをテキスト形式で表した文字列を返します。

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100',
  'insensitive', 'read-only' );
```

EXPRTYPE 関数 [その他]

式のデータ型を識別する文字列を返します。

構文

EXPRTYPE(*string-expression*, *integer-expression* **)**

パラメーター

- ***string-expression***　SELECT 文。式のデータ型が問い合わされる場合、その式は SELECT リストに表示されます。文字列が有効な SELECT 文でない場合は、NULL を返します。

- ***integer-expression***　対象となる式の SELECT リストでの位置。SELECT リストの最初の項目が 1 番になります。integer-expression 値が SELECT リストの項目に対応しない場合は、NULL を返します。

戻り値

LONG VARCHAR

備考

ユーザー定義のドメインでは、EXPRTYPE はドメイン名ではなく基本となるデータ型の説明を返します。たとえば、次のように指定して mydomain というドメインを作成し、mydomain を使用してテーブルカラムを定義するとします。

```
CREATE DOMAIN mydomain CHAR(20);
CREATE TABLE mytable( colA mydomain, colB DATETIME );
```

SELECT EXPRTYPE('SELECT * FROM mytable', 1) を実行すると、返されるデータ型は char(20) になり、mydomain ではありません。

参照

- 「SQL データ型」85 ページ
- 「sa_describe_query システムプロシージャー」1074 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、SQL Anywhere サンプルデータベースに対して実行した場合に、smallint を返します。

```
SELECT EXPRTYPE( 'SELECT LineID FROM SalesOrderItems', 1 );
```

FIRST_VALUE 関数 [集合]

ウィンドウの最初のローの値を返します。

構文

FIRST_VALUE([ALL] *expression* **[{ RESPECT | IGNORE } NULLS])**
OVER (*window-spec* **)**

window-spec :次の「備考」の項を参照してください

パラメーター

- ***expression***　評価する式。たとえば、カラム名です。

戻り値

ウィンドウの最初のローの値のデータ型。

備考

FIRST_VALUE 関数を使用すると、セルフジョインを使用せずに、(何らかの順序による) 最初の値を選択できます。最初の値を計算の基準として使用する場合、この関数が役立ちます。

FIRST_VALUE 関数は、ウィンドウの最初のレコードを取得します。次に、最初のレコードに対して *expression* が比較され、結果が返されます。

IGNORE NULLS を指定すると、*expression* にある最初の NULL 以外の値が返されます。RESPECT NULLS (デフォルト) を指定すると、最初の値が、それが NULL であってもなくても返されます。

FIRST_VALUE 関数は、その他の大部分の集合関数とは異なり、ウィンドウ指定を行った場合にのみ使用できます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

window-spec の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。詳細については、WINDOW 句の *window-spec* 定義を参照してください。「WINDOW 句」989 ページを参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「Window 集合関数」『SQL Anywhere サーバー SQL の使用法』
- 「LAST_VALUE 関数 [集合]」262 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

  SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」がサポートされています。これにより、カラム参照ではない任意の式を Window 関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、FIRST_VALUE 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例は、各従業員の給料と、同じ部署内で最近採用された従業員の給料との関係を、パーセンテージで返します。

```
SELECT DepartmentID, EmployeeID,
    100 * Salary / ( FIRST_VALUE( Salary ) OVER (
                PARTITION BY DepartmentID  ORDER BY StartDate DESC ) )
      AS percentage
  FROM Employees;
```

DepartmentID	EmployeeID	percentage
500	1658	100
500	1615	110.4284624
500	1570	138.8427097
500	1013	109.5851905
500	921	167.4497049

DepartmentID	EmployeeID	percentage
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
400	1607	175.1828989
400	1576	197.0164609
...	...	...

従業員 1658 は部署 500 の最初のローに示されていることから、従業員 1658 がこの部署で最近採用された従業員であることがわかります。パーセンテージは 100% に設定されています。部署 500 の他の従業員のパーセンテージは、従業員 1658 との相対的な値として算出されます。たとえば、従業員 1570 は、従業員 1658 の給料の約 139% に相当する給料を受け取っています。

同じ部署内にいるその他の従業員が、最近採用された従業員と同じ額の給料を受け取っている場合は、その従業員のパーセンテージも 100 になります。

FLOOR 関数 [数値]

指定された値以下の、最大の整数値を返します。

構文

FLOOR(*numeric-expression* **)**

パラメーター

- ***numeric-expression*** トランケートされる値。通常は、位取りがゼロ以外の固定数値型または適切な数値型 (DOUBLE、REAL、または FLOAT) です。

戻り値

DOUBLE

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

参照

● 「CEILING 関数 [数値]」 166 ページ

標準と互換性

● **SQL/2008**　FLOOR 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の文は、123 の Floor 値を返します。

```
SELECT FLOOR (123);
```

次の文は、値 123 を返します。

```
SELECT FLOOR (123.45);
```

次の文は、値 -124 を返します。

```
SELECT FLOOR (-123.45);
```

GET_BIT 関数 [ビット配列]

ビット配列の指定したビットの値 (1 または 0) を返します。

構文

GET_BIT(*bit-expression*, *position* **)**

パラメーター

● ***bit-expression***　ビットを含むビット配列。

● ***position***　ステータスを返すビットの位置。

戻り値

BIT

備考

配列の位置は左側からカウントします。初期値は 1 です。

position は配列の長さを超える場合、0 (false) が返されます。

参照

- 「ビット処理演算子」 11 ページ
- 「SET_BIT 関数 [ビット配列]」 341 ページ
- 「SET_BITS 関数 [集合]」 343 ページ
- 「sa_get_bits システムプロシージャー」 1086 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 1 を返します。

```
SELECT GET_BIT( '00110011' , 4 );
```

次の文は、値 0 を返します。

```
SELECT GET_BIT( '00110011' , 5 );
```

GET_IDENTITY 関数 [その他]

AUTOINCREMENT カラムに値を割り当てます。AUTOINCREMENT を使用して数を生成する代わりに、この関数を使用できます。

構文

GET_IDENTITY(*table_name* **[,** *number_to_allocate* **])**

パラメーター

- ***table_name***　テーブルの名前を指定する文字列。オプションで所有者名を含みます。
- ***number_to_allocate***　予約する値の数。デフォルトは 1 です。

戻り値

UNSIGNED BIGINT

備考

最も効率的な ID の生成方法は AUTOINCREMENT または GLOBAL AUTOINCREMENT を使用する方法ですが、この関数は代替手段として提供されています。この関数は、テーブルに AUTOINCREMENT カラムが定義されていることを前提としています。テーブルの AUTOINCREMENT カラムの生成に使用可能な次の値を返し、他の接続がデフォルトでその値を使用しないように値を予約します。

テーブルが見つからない場合はエラーを返し、テーブルに AUTOINCREMENT カラムが存在しない場合は NULL を返します。複数の AUTOINCREMENT カラムがある場合は、最初に検出されたものが使用されます。

number_to_allocate は予約する値の数です。*number_to_allocate* に 1 より大きい値を指定すると、残りの値も予約されます。次回の割り当てでは、現在の数に *number_to_allocate* の値を加算した

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

数が使用されます。これによって、アプリケーションは GET_IDENTITY 関数を頻繁に実行せずにすみます。*number_to_allocate* が 0 の場合は、値が予約されずに、次に使用可能な値が返されます。

GET_IDENTITY 関数の実行後は COMMIT が不要のため、ローの挿入に使用するのと同じ接続を使用して呼び出すことができます。例に示すように、いくつかのテーブルの ID 値が必要な場合は、GET_IDENTITY 関数の呼び出しを複数持つ単一の SELECT を使用して取得できます。

GET_IDENTITY 関数は、非決定的関数です。以降、GET_IDENTITY 関数を呼び出すと異なる値が返される可能性があります。オプティマイザーは、GET_IDENTITY 関数の結果をキャッシュしません。

非決定的関数の詳細については、「関数のキャッシュ」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「CREATE TABLE 文」651 ページ
- 「ALTER TABLE 文」462 ページ
- 「NUMBER 関数 [その他]」296 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、Customers テーブルの AUTOINCREMENT カラム (ID) に対して次に使用できる数値を返します。返された値とその後の 9 個の値が予約されます。

```
SELECT GET_IDENTITY( 'Customers', 10 );
```

GETDATE 関数 [日付と時刻]

現在の年、月、日、時、分、秒、秒以下を返します。

構文

GETDATE( )

戻り値

TIMESTAMP

備考

精度はシステムクロックの精度によって制限されます。

GETDATE 関数が返す情報は、NOW 関数と CURRENT TIMESTAMP 特別値が返す情報と同じです。

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「式」12 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、システムの日付と時刻を返します。

```
SELECT GETDATE( );
```

GRAPHICAL_PLAN 関数 [その他]

SQL 文のプランの最適化方法を、XML フォーマットの文字列で返します。

構文

GRAPHICAL_PLAN(
string-expression
[, *statistics-level*
[, *cursor-type*
[, *update-status*]]] **)**

パラメーター

- ***string-expression***　SQL 文。通常は SELECT 文ですが、UPDATE 文または DELETE 文も指定できます。

- ***statistics-level***　整数。*Statistics-level* には、次のいずれかの値を指定します。

値	説明
0	オプティマイザーの推定のみ (デフォルト)

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

値	説明
2	ノード統計値を含む詳細な統計情報
3	詳細な統計情報

- ***cursor-type*** カーソルタイプ。文字列として表現されます。使用できる値は、asensitive、insensitive、sensitive、または keyset-driven です。*cursor-type* が指定されない場合、デフォルトで asensitive が使用されます。

- ***update-status*** 次のいずれかの値を受け入れる文字列パラメーター。これらの値は、指定されたカーソルをオプティマイザーがどのように処理するかを示します。

値	説明
READ-ONLY	このカーソルは読み込み専用。
READ-WRITE (デフォルト)	このカーソルは読み込みや書き込みが可能。
FOR UPDATE	このカーソルは読み込みや書き込みが可能。READ-WRITE とまったく同じです。

戻り値

LONG VARCHAR

参照

- 「高度：クエリ実行プラン」『SQL Anywhere サーバー SQL の使用法』
- 「PLAN 関数 [その他]」302 ページ
- 「EXPLANATION 関数 [その他]」228 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の Interactive SQL の例は、SELECT 文を文字列パラメーターとして渡し、クエリを実行するためのプランを返します。プランは *plan.saplan* ファイルに保存されます。これは、Interactive SQL を使用して読み込むことができます。

```
SELECT GRAPHICAL_PLAN( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
OUTPUT TO 'plan.saplan' FORMAT TEXT QUOTE '' HEXADECIMAL ASIS;
```

次の文は、SELECT * FROM Departments WHERE DepartmentID > 100 クエリに対するキーセット駆動型の更新可能なカーソルのグラフィカルなプランを含む文字列を返します。また、オプティマイザーが使用した推定統計情報に加え、実際の実行の統計情報の注釈がサーバーによってプランに付けられます。

```
SELECT GRAPHICAL_PLAN(
  'SELECT * FROM Departments WHERE DepartmentID > 100',
  2,
  'keyset-driven', 'for update' );
```

GREATER 関数 [その他]

2 つのパラメーター値のうち、より大きい値を返します。

構文

GREATER(*expression-1*, *expression-2* **)**

パラメーター

- ***expression-1*** 比較される最初のパラメーター値。
- ***expression-2*** 比較される 2 番目のパラメーター値。

戻り値

この関数の戻り値は、指定した式によって異なります。具体的には、データベースサーバーが関数を評価するとき、まず、式の比較が可能なデータ型を検索します。該当するデータ型が見つかると、データベースサーバーは式を比較し、比較に使用したデータ型で結果を返します。データベースサーバーは、一般に比較が可能なデータ型を見つけることができないと、エラーを返します。

備考

パラメーターが等しい場合は、最初のパラメーターを返します。

参照

- 「LESSER 関数 [その他]」267 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 10 を返します。

```
SELECT GREATER( 10, 5 ) FROM dummy;
```

GROUPING 関数 [集合]

GROUP BY 演算の結果セット内のカラムが NULL である場合、その理由が小計ローの一部であるためか、または基本データによるためかを識別します。

構文

GROUPING(*group-by-expression* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***group-by-expression*** GROUP BY 句を使用するクエリの結果セット内において、グループ化カラムとして表示される式。この関数を使用して、ROLLUP 演算または CUBE 演算で設定された結果セットに追加される小計ローを調べることができます。

戻り値

- **1** 小計ローの一部なので、*group-by-expression* が NULL であることを示します。カラムは、そのローのプレフィクスカラムではありません。

- **0** 小計ローのプレフィクスカラムなので、*group-by-expression* が NULL であることを示します。

参照

- 「ROLLUP 句」『SQL Anywhere サーバー SQL の使用法』
- 「CUBE 句」『SQL Anywhere サーバー SQL の使用法』
- 「GROUP BY GROUPING SETS」『SQL Anywhere サーバー SQL の使用法』
- 「SELECT 文」895 ページ
- 「GROUPING 関数を使用した NULL の検出」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** GROUPING 関数は、オプションの SQL/2008 言語機能 T431、「Extended grouping capabilities」の一部です。

例

この関数の使用例については、「GROUPING 関数を使用した NULL の検出」『SQL Anywhere サーバー SQL の使用法』を参照してください。

HASH 関数 [文字列]

指定された値をハッシュ形式で返します。

構文

HASH(*string-expression*[, *algorithm*] **)**

パラメーター

- ***string-expression*** ハッシュされる文字列。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

- ***algorithm*** ハッシュに使用するアルゴリズム。可能な値には CRC32、MD5、SHA1、SHA1_FIPS、SHA256、SHA256_FIPS があります。デフォルトでは、MD5 アルゴリズムが使用されます。FIPS 認定のアルゴリズムには別のライセンスが必要です。

戻り値

使用するアルゴリズムごとに返される型を次に示します。

- CRC32 は 16 進数文字列を返します。16 進数文字列を 32 ビット整数に変換するには、HEXTOINT 関数を使用します。
- MD5 は VARCHAR(32) を返します
- SHA1 は VARCHAR(40) を返します
- SHA1_FIPS は VARCHAR(40) を返します
- SHA256 は VARCHAR(64) を返します
- SHA256_FIPS は VARCHAR(64) を返します

備考

ハッシュを使用すると、値は、関数に渡されたそれぞれの値に対してユニークなバイトシーケンスに変換されます。

データベースサーバーを -fips オプションを使用して起動すると、使用されるアルゴリズムや動作が次のように異なる場合があります。

- SHA1 を指定する場合、SHA1_FIPS が使用されます。
- SHA256 を指定する場合、SHA256_FIPS が使用されます。
- MD5 を指定する場合、エラーが返されます。
- FIPS 認定ではない CRC32 アルゴリズムは暗号化アルゴリズムとして認識されないため、FIPS で使用できます。

警告
すべてのアルゴリズムは一方向のハッシュです。ハッシュから元の文字列を再作成することはできません。

参照

- 「文字列関数」148 ページ
- 「-fips dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「SQL Anywhere のセキュリティオプション」『SQL Anywhere 12 紹介』
- 「HEXTOINT 関数 [データ型変換]」241 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、ユーザー ID やパスワードなど、アプリケーションのユーザーに関する情報を格納するテーブル user_info を作成します。テーブルにはローが 1 つ挿入されます。パスワードは、HASH 関数と SHA256 アルゴリズムを使用してハッシュされます。この方法でハッシュ済みパスワードを格納する方法は、クリアテキストでパスワードを格納せず、パスワードの比較を必要とする外部アプリケーションがある場合に有効です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
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' ) );
```

HEXTOINT 関数 [データ型変換]

16 進文字列と同等の 10 進整数を返します。

CAST、CONVERT、HEXTOINT、INTTOHEX 関数を使用すると、16 進値変換を行うことができます。これらの関数の使用の詳細については、「16 進値との変換」6 ページを参照してください。

構文

HEXTOINT(*hexadecimal-string* **)**

パラメーター

- ***hexadecimal-string*** 整数に変換される文字列。

戻り値

HEXTOINT 関数は、プラットフォームに依存しない SQL INTEGER に相当する 16 進文字列を INT として返します。右から 8 桁目が数値 8 ～ 9 か、大文字または小文字の A ～ F のいずれかであり、その前の桁がすべて大文字または小文字の F の場合は、16 進値が負の整数値になります。次の HEXTOINT は無効な使用例です。これは、引数が符号付き 32 ビット整数で表現できない正の整数値を示しているためです。

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

備考

HEXTOINT 関数は、数値、大文字または小文字の A ～ F のみで構成される文字列リテラルまたは変数を受け入れます (0x プレフィクスが付いている場合、付いていない場合の両方)。次に HEXTOINT の有効な使用例をすべて示します。

```
SELECT HEXTOINT( '0xFFFFFFFF' );
SELECT HEXTOINT( '0x00000100' );
SELECT HEXTOINT( '100' );
SELECT HEXTOINT( '0xffffffff80000001' );
```

HEXTOINT 関数は 0x プレフィクスがあれば削除します。データが 8 桁を超える場合、符号付き 32 ビット整数値として表現できる値を示す必要があります。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「INTTOHEX 関数 [データ型変換]」257 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 420 を返します。

```
SELECT HEXTOINT( '1A4' );
```

HOUR 関数 [日付と時刻]

TIMESTAMP 値の時間部分を返します。

構文

HOUR(*timestamp-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP 値。

戻り値

SMALLINT

備考

TIMESTAMP 式の時間部分である、0 ～ 23 の間の SMALLINT 値が返されます。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 21 を返します。

```
SELECT HOUR( '1998-07-09 21:12:13' );
```

HOURS 関数 [日付と時刻]

HOURS 関数は、TIMESTAMP を操作するか、2 つの TIMESTAMP 値の間の時間数を返します。詳細については、この関数の使用法を参照してください。

構文 1

HOURS (*timestamp-expression* **)**

構文 2

HOURS (*timestamp-expression*, *timestamp-expression* **)**

構文 3

HOURS (*time-or-timestamp-expression*, *integer-expression* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***time-or-timestamp-expression*** TIME または TIMESTAMP データ型の値。
- ***timestamp-expression*** TIMESTAMP データ型の値。
- ***integer-expression*** *time-or-timestamp-expression* に加算される時間数。*integer-expression* が負の場合、*time-or-timestamp-expression* から適切な時間数が引かれます。

  データ型のキャストの詳細については、「CAST 関数 [データ型変換]」165 ページを参照してください。

戻り値

構文 1 または構文 2 では INTEGER。

構文 3 では TIME または TIMESTAMP。

備考

HOURS 関数の結果は、その引数によって異なります。

- **構文 1** 1 つの *timestamp-expression* を HOURS 関数に渡すと、0000-02-29 の真夜中から *timestamp-expression* までの時間数を INTEGER として返します。

  > **注意**
  > 0000-02-29 は実際の日付を指すための値ではありません。HOURS 関数で使用されるデフォルトの TIMESTAMP 値です。

- **構文 2** 2 つの TIMESTAMP 値を HOURS 関数に渡すと、その間の時間数を整数で返します。
- **構文 3** TIMESTAMP 値と INTEGER 値を HOURS 関数に渡すと、整数の時間数を *time-or-timestamp-expression* 引数に加算した TIMESTAMP 結果を返します。同様に、TIME 値を最初の引数として渡すと、TIME 値を結果として返します。構文 3 では、最初の引数の暗黙的変換をサポートしていません。最初の引数を DATE、TIME または TIMESTAMP の値に明示的にキャストしてください。最初の引数が DATE の場合、時刻部分には真夜中が使用されます。

構文 2 の代わりに、DATEDIFF 関数を使用します。構文 3 の代わりに、DATEADD 関数を使用します。

参照

- 「DATEDIFF 関数 [日付と時刻]」196 ページ
- 「DATEADD 関数 [日付と時刻]」194 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 4 を返します。これは、2 番目の TIMESTAMP 値が、最初のタイムスタンプの 4 時間後であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。

```
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' );
```

次の文は、値 17517342 を返します。

```
SELECT HOURS( '1998-07-13 06:07:12' );
```

次の文は、日時 1999-05-13 02:05:07.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT HOURS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );
SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );
```

HTML_DECODE 関数 [その他]

HTML リテラル文字列で表示される特殊文字エンティティを復号化します。

構文

HTML_DECODE(*string* **)**

パラメーター

- ***string*** HTML ドキュメントで使用される任意のリテラル文字列。

戻り値

LONG VARCHAR または LONG NVARCHAR

備考

この関数は、適切な置換を行った後に文字列引数を返します。次の表は、有効な文字エンティティのサンプルを示します。

文字	置換
"	"
'	'
&	&
<	<
>	>

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

文字	置換
&#x*hexadecimal-number*;	Unicode のコードポイント。16 進数で指定します。たとえば、' は一重引用符を返します。
&#*decimal-number*;	Unicode のコードポイント。10 進数で指定します。たとえば、™ は商標記号を返します。

指定した Unicode のコードポイントが、データベース側文字セットの文字に変換できる場合は、その文字に変換されます。それ以外の場合は、解釈されないまま返されます。

SQL Anywhere は、HTML 4.01 仕様で指定されているすべての文字エンティティ参照をサポートします。http://www.w3.org/TR/html4/ と http://www.w3.org/TR/html4/sgml/entities.html#h-24.2 を参照してください。

参照

- 「HTML_ENCODE 関数 [その他]」245 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

```
SELECT HTML_DECODE('&lt;p&gt;The piano was made ' ||
  'by &lsquo;Steinway &amp; Sons&rsquo;.&lt;/p&gt;')

SELECT HTML_DECODE('&lt;p&gt;It cost &euro;85.000,00.&lt;/p&gt;')
```

HTML_ENCODE 関数 [その他]

HTML ドキュメントに挿入する文字列内の特殊文字をエンコードします。

構文

HTML_ENCODE(*string* **)**

パラメーター

- ***string***　HTML ドキュメントで使用される任意の文字列。

戻り値

LONG VARCHAR または LONG NVARCHAR

備考

この関数は、次の一連の置換を行った後に文字列引数を返します。

文字	置換
"	"
'	'
&	&
<	<
>	>
0x20 より小さいコード *nn*	&#x*nn*;

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「HTML_DECODE 関数 [その他]」244 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、文字列 '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"> ' を返します。

```
SELECT HTML_ENCODE('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">')
```

HTTP_BODY 関数 [Web サービス]

HTTP 要求の本文をバイナリ形式で返します。たとえば、POST 要求では、これは未加工の POST データになります。

構文

HTTP_BODY( )

パラメーター

なし

戻り値

HTTP 要求の本文を含む LONG VARCHAR をバイナリ形式で返し、文字セット変換は行われません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

要求の本文が存在しない場合、または関数が Web サービスから呼び出されない場合は、NULL 値が返されます。

この関数は、PHP 外部環境で便利です。

参照

- 「sa_http_php_page システムプロシージャー」1098 ページ
- 「sa_http_php_page_interpreted システムプロシージャー」1099 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

HTTP_DECODE 関数 [Web サービス]

HTTP のコード化された文字列を復号化します。これは URL 復号化とも呼ばれます。

構文

HTTP_DECODE(*string* **)**

パラメーター

- ***string*** URL またはコード化された URL の要求本文から取得された任意の文字列。

戻り値

LONG VARCHAR または LONG NVARCHAR

備考

この関数は、%*nn* という形式 (*nn* は 16 進値) のすべての文字シーケンスをコード *nn* の文字で置換した後に、文字列引数を返します。また、すべてのプラス記号 (+) はスペースで置換されます。

参照

- 「HTTP_ENCODE 関数 [Web サービス]」248 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

```
SELECT HTTP_DECODE('http%3A%2F%2Fdcx.sybase.com')
```

HTTP_ENCODE 関数 [Web サービス]

HTTP で使用するために文字列をコード化します。これは URL コード化とも呼ばれます。

構文

HTTP_ENCODE(*string* **)**

パラメーター

- ***string*** HTTP トランスポート用にコード化される任意の文字列。

戻り値

LONG VARCHAR または LONG NVARCHAR

備考

この関数は、次の一連の置換を行った後に文字列引数を返します。また、16 進コードが 20 より小さいか 7E より大きいすべての文字は、%*nn* (*nn* は文字コード) で置換されます。

文字	置換
スペース	%20
"	%22
#	%23
%	%25
&	%26
,	%2C
;	%3B
<	%3C
>	%3E
[	%5B
¥	%5C
]	%5D
`	%60
{	%7B

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

文字	置換
|	%7C
}	%7D
0x20 未満で 0x7f を超える文字コード *nn*	%*nn*

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「HTTP_DECODE 関数 [Web サービス]」247 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

```
SELECT HTTP_ENCODE('/opt&id=123&text="oid:c¥x09d ef"')
```

HTTP_HEADER 関数 [Web サービス]

HTTP 要求ヘッダーの値を返します。

構文

HTTP_HEADER(*header-field-name* **)**

パラメーター

- ***header-field-name***　HTTP 要求ヘッダーフィールドの名前。

戻り値

LONG VARCHAR

備考

この関数は、指定された HTTP 要求ヘッダーフィールドの値を返します。ヘッダーが存在しない場合、または HTTP サービスから呼び出されていない場合は NULL を返します。Web サービスを介して HTTP 要求を処理する場合に使用します。

HTTP Web サービス要求を処理するとき、次のようなヘッダーが有用です。

- **Cookie**　要求された URI に関連付けられた cookie 値 (クライアントに格納されている場合)。
- **Referer**　要求された URI へのリンクが指定されたページの URL。

- **Host**　要求を送信したホストの名前または IP。
- **User-Agent**　クライアントアプリケーション名。
- **Accept-Encoding**　クライアントアプリケーションが使用できる応答のエンコーディングリスト。

これらのヘッダーの詳細については、http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html を参照してください。

次の特別なヘッダーでは、クライアント要求の要求行で要素にアクセスできます。

- **@HttpMethod**　処理されている要求の種類を返します。可能な値には DELETE、HEAD、GET、PUT、POST があります。
- **@HttpURI**　HTTP 要求で指定された、要求の完全な URI (/myservice?&id=-123&version=109&lang=en など)。
- **@HttpVersion**　要求の HTTP バージョン (HTTP/1.0 または HTTP/1.1 など)。
- **@HttpQueryString**　要求された URI が存在する場合に、そのクエリ部分を返します (&id=-123&version=109&lang=en など)。

参照

- 「NEXT_HTTP_HEADER 関数 [Web サービス]」290 ページ
- 「sa_set_http_header システムプロシージャー」1185 ページ
- 「sa_http_header_info システムプロシージャー」1097 ページ
- 「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は Cookie ヘッダーの値を取得します。

```
SET cookie_value = HTTP_HEADER( 'Cookie' );
```

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は、HTTP 要求ヘッダーの名前と値をデータベースサーバーメッセージウィンドウに表示します。

```
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;
```

HTTP_RESPONSE_HEADER 関数 [Web サービス]

HTTP 応答ヘッダーの値を返します。

構文

HTTP_RESPONSE_HEADER(*header-field-name* **)**

パラメーター

- ***header-field-name*** HTTP 応答ヘッダーフィールドの名前。

戻り値

LONG VARCHAR

備考

この関数は、指定された HTTP 応答ヘッダーフィールドの値を返します。指定された *header-field-name* のヘッダーが存在しない場合、または HTTP サービスから呼び出されていない場合は NULL を返します。

HTTP Web サービス応答を処理するとき、次のようなヘッダーが有用です。

- **Connection** [Connection] フィールドで送信者が特定の接続に必要なオプションを指定できます。SQL Anywhere HTTP サーバー応答では、このオプションは常に「閉じる」です。
- **Content-Length** [Content-Length] フィールドには、応答本文のサイズが 10 進数のオクテットで表示されます。
- **Content-Type** [Content-Type] フィールドには、受信者に送信される本文のメディアタイプが表示されます。例： **text/xml**
- **Date** [Date] フィールドは、応答が発信された日付と時刻を表します。
- **Expires** [Expires] フィールドには、それ以降、応答が古くなったとみなされる日付と時刻が表示されます。
- **Location** [Location] フィールドは、受信者を新しいリソースの要求または識別を完了するロケーションにリダイレクトします。

- **Server**　[Server] フィールドには元のサーバーで要求を処理するために使用されるソフトウェアに関する情報が含まれます。SQL Anywhere HTTP サーバー応答では、Web サーバー名がバージョン番号と一緒に表示されます。

- **Transfer-Encoding**　[Transfer-Encoding] フィールドには、送信者と受信者の間でメッセージを安全に転送するためにメッセージ本文に適用される変換があれば、その変換の種類が表示されます。

- **User-Agent**　[User-Agent] フィールドには、要求の発信元のユーザーエージェントに関する情報が表示されます。SQL Anywhere HTTP サーバー応答では、Web サーバー名がバージョン番号と一緒に表示されます。

- **WWW-Authenticate**　[WWW-Authenticate] フィールドは、401 (認証されていない) 応答メッセージに含まれます。

これらのヘッダーの詳細については、http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html を参照してください。

次の特別なヘッダーでは、サーバー応答の応答でステータスにアクセスできます。

- **@HttpStatus**　処理された要求のステータスコードを返します。

参照

- 「NEXT_HTTP_RESPONSE_HEADER 関数 [Web サービス]」291 ページ
- 「sa_set_http_header システムプロシージャー」1185 ページ
- 「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は、HTTP 応答ヘッダーの名前と値をデータベースサーバーメッセージウィンドウに表示します。

```
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;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

HTTP_VARIABLE 関数 [Web サービス]

HTTP 変数の値を返します。

構文

HTTP_VARIABLE(*var-name* [, *instance* [, *attribute*]] **)**

パラメーター

- ***var-name*** HTTP 変数の名前。
- ***instance*** 同じ名前の変数が複数ある場合、フィールドインスタンスのインスタンス番号、または最初の NULL。複数選択を許可している SELECT リストで使用すると便利です。
- ***attribute*** マルチパート要求では、マルチパート名のヘッダーの値を返すヘッダーフィールド名を属性で指定できます。

  属性を指定しないと、戻り値が % で復号化され、文字セットがデータベースの文字セットに変換されます。このモードでは、UTF % でコード化されたデータがサポートされています。

  属性には、次のいずれかのモードも使用できます。

  - **'@BINARY'** x-www-form-urlencoded バイナリデータ値を返します。このモードは、戻り値が % で復号化されており、変換された文字セットではないことを示します。UTF-8 % エンコードは、このモードではサポートされません。% コード化されたデータは単純に等価のバイト表現にデコードされるからです。
  - **'@TRANSPORT'** 未加工の HTTP トランスポート形式の値を返し、% でのコード化が保持されます。

戻り値

LONG VARCHAR

備考

この関数は、指定された HTTP 変数の値を返します。Web サービス内で HTTP 要求を処理する場合に使用されます。

var-name が存在しない場合、戻り値は NULL になります。

Web サービス要求が POST で、変数データが multipart/form-data と通知された場合、HTTP サーバーは個々の変数の HTTP ヘッダーを受信します。*attribute* パラメーターを指定すると、特定の変数の POST 要求から関連する multipart/form-data ヘッダー値が HTTP_VARIABLE 関数から返されます。ファイルを表す変数の場合、Content-Disposition、Content-Type、@BINARY の属性はそれぞれファイル名、メディアタイプ、ファイルの内容を返します。

通常は、クライアント (ブラウザーなど) の文字セットとデータベース側文字セットの間で、どの入力データも文字セットの変換が行われます。ただし、*attribute* に @BINARY を指定すると、文字セットの変換または % による復号化を行わずに変数値が返されます。@BINARY は、画像などのデータをクライアントから受信するときに便利です。

この関数は、指定されたインスタンスが存在しない場合、または Web サービスの実行外から関数が呼び出された場合には、NULL を返します。

参照

- 「NEXT_HTTP_VARIABLE 関数 [Web サービス]」292 ページ
- 「sa_http_variable_info システムプロシージャー」1100 ページ
- 「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は、サンプル URL で示される HTTP 変数の値を取得します。

```
-- 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;
```

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の文はイメージ変数の Content-Disposition ヘッダーと Content-Type ヘッダーを要求します。

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
```

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の文は現在の文字セットでのイメージ変数値 (つまり、文字セットを変換しない値) を要求します。

```
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

IDENTITY 関数 [その他]

クエリの連続した各ローに対して、1 から開始する整数値を生成します。この関数の実装は、NUMBER 関数の場合と同じです。

構文

IDENTITY(*expression* **)**

パラメーター

- ***expression***　式。この式は解析されますが、関数の実行時には無視されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

INT

備考

IDENTITY 関数の説明は NUMBER 関数の説明と同じです。

参照

- 「NUMBER 関数 [その他]」296 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、連番が付けられた従業員リストを返します。

```
SELECT IDENTITY( 10 ), Surname FROM Employees;
```

IFNULL 関数 [その他]

最初の式が NULL 値の場合は、2 番目の式の値を返します。最初の式が NULL でない場合は、3 番目の式の値を返します。最初の式が NULL ではなく、3 番目の式が存在しない場合は、NULL を返します。

構文

IFNULL(*expression-1*, *expression-2* [, *expression-3*] **)**

パラメーター

- ***expression-1*** 評価される式。この値によって、*expression-2* または *expression-3* のどちらが返るかが決まります。
- ***expression-2*** *expression-1* が NULL の場合の戻り値。
- ***expression-3*** *expression-1* が NULL でない場合の戻り値。

戻り値

返されるデータ型は、*expression-2* と *expression-3* のデータ型によって異なります。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 -66 を返します。

```
SELECT IFNULL( NULL, -66 );
```

次の文は、最初の式が NULL ではなく、3 番目の式が存在しないため、NULL を返します。

```
SELECT IFNULL( -66, -66 );
```

INDEX_ESTIMATE 関数 [その他]

指定したパラメーターに基づいてクエリオプティマイザーによって計算されたパーセンテージとして、インデックスの選択性推定を返します。

構文

INDEX_ESTIMATE(*column-name* **[,** *value* **[,** *relation-string* **]])**

パラメーター

- ***column-name*** 推定で使用されるカラム。
- ***value*** カラムが比較される値。デフォルト値は NULL です。
- ***relation-string*** 比較に使用される比較演算子。一重引用符で囲んで指定します。このパラメーターに使用できる値は、'='、'>'、'<'、'>='、'<='、'<>'、'!='、'!<'、'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '!>' です。デフォルトは '=' です。

戻り値

REAL

備考

この関数は、述部 column-name relation-string value のインデックスから選択性推定を返します。*value* が NULL で、比較演算子が '=' の場合は、述部 column-name IS NULL の選択性推定になります。*value* が NULL で、比較演算子が '!=' または '<>' の場合は、述部 column-name IS NOT NULL の選択性推定になります。

参照

- 「ESTIMATE 関数 [その他]」219 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」220 ページ
- 「EXPERIENCE_ESTIMATE 関数 [その他]」227 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、200 より大きいと推定される EmployeeID 値のパーセンテージを返します。

```
SELECT INDEX_ESTIMATE( EmployeeID, 200, '>' )
FROM Employees;
```

INSERTSTR 関数 [文字列]

別の文字列の指定された位置に文字列を挿入します。

構文

INSERTSTR(*integer-expression*, *string-expression-1*, *string-expression-2* **)**

パラメーター

- ***integer-expression*** 文字列の挿入位置。文字列の先頭に挿入する場合は、0 を使用します。
- ***string-expression-1*** 別の文字列が挿入される文字列。
- ***string-expression-2*** 挿入する文字列。

戻り値

LONG VARCHAR

備考

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「STUFF 関数 [文字列]」361 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 backoffice を返します。

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```

INTTOHEX 関数 [データ型変換]

整数に対応する 16 進値の文字列を返します。

構文

INTTOHEX(*integer-expression* **)**

パラメーター

- ***integer-expression*** 16 進に変換される整数。

戻り値

VARCHAR

備考

CAST、CONVERT、HEXTOINT、INTTOHEX 関数を使用すると、16 進値変換を行うことができます。詳細については、「16 進値との変換」6 ページを参照してください。

参照

- 「HEXTOINT 関数 [データ型変換]」241 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 0000009c を返します。

```
SELECT INTTOHEX( 156 );
```

ISDATE 関数 [データ型変換]

文字列引数を日付に変換できるかどうかをテストします。

構文

ISDATE(*string* **)**

パラメーター

- ***string***　文字列が有効な日付を表すかどうかを判断するために分析される文字列。

戻り値

INT

備考

変換できる場合は 1 を返し、できない場合は 0 を返します。引数が NULL の場合は 0 を返します。

この関数は NCHAR の入力または出力をサポートしています。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「NOW 関数 [日付と時刻]」294 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、外部ファイルからデータをインポートし、無効な値があるローをエクスポートし、残りのローを永久テーブルにコピーします。

```
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;
```

ISENCRYPTED 関数 [システム]

ENCRYPT 関数および指定のキーを使用して文字列が暗号化されているかどうかを判断します。

構文

ISENCRYPTED(*string*, *key*[, *algorithm*] **)**

戻り値

INT

パラメーター

- ***string*** 暗号化されているかどうかを判断するために分析される文字列。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

- ***key*** *string* の暗号化に使用される暗号化キー。大文字と小文字を区別しないデータベース定義であっても、パラメーターの大文字と小文字は区別されます。

- ***algorithm*** このオプションのパラメーターは、*string* の暗号化に使用されたアルゴリズムを指定します。サポートされているアルゴリズムは、AES、AES256、AES_FIPS、および AES256_FIPS です。

  FIPS 認定の暗号化をサポートしているプラットフォームでは、*algorithm* に FIPS 認定のアルゴリズムのいずれかを指定できます。

  > **注意**
  > RSA、ECC、FIPS 認定の各暗号化は、すべてのプラットフォームで使用できるわけではありません。各プラットフォームでサポートされている暗号化方法については、http://www.ianywhere.jp/tech/1061806-os_components.html を参照してください。

  algorithm が指定されない場合、デフォルトで AES が使用されます。データベースサーバーの起動時に -fips サーバーオプションが使用された場合、デフォルトは AES_FIPS になります。

備考

ISENCRYPTED は、入力文字列が指定のキーを使用して暗号化されている場合は 1 を返し、それ以外の場合は 0 を返します。

参照

- 「ENCRYPT 関数 [文字列]」216 ページ
- 「DECRYPT 関数 [文字列]」211 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次のフラグメントは、ISENCRYPTED 関数の使い方を示します。

```
SELECT ISENCRYPTED( ENCRYPT ('test_string', 'key' ), 'key');
```

ISNULL 関数 [その他]

リストの中から NULL でない最初の式を返します。この関数は COALESCE 関数と同じです。

構文

ISNULL(*expression*, *expression* [, ...] **)**

パラメーター

- ***expression*** NULL かどうかがテストされる式。

  2 つ以上の式を関数に渡します。すべての式は比較可能であることが必要です。

戻り値

この関数の戻り値は、指定した式によって異なります。具体的には、データベースサーバーが関数を評価するとき、まず、式の比較が可能なデータ型を検索します。該当するデータ型が見つかると、データベースサーバーは式を比較し、比較に使用したデータ型で結果を返します。データベースサーバーは、一般に比較が可能なデータ型を見つけることができないと、エラーを返します。

参照

- 「COALESCE 関数 [その他]」 170 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 -66 を返します。

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 );
```

ISNUMERIC 関数 [その他]

文字列の引数が有効な数値であるかどうかを判断します。

構文

ISNUMERIC(*string* **)**

パラメーター

- ***string*** 文字列が有効な数値を表すかどうかを判断するために分析される文字列。

戻り値

INT

備考

ISNUMERIC は、入力文字列が有効な整数または浮動小数点値であると評価された場合は 1、それ以外の場合は 0 を返します。また、文字列に空白のみが含まれているか、または NULL である場合も 0 を返します。

次の例も、ISNUMERIC 関数から 0 を返します。

●文字 d または D を指数セパレーターに使用する値。たとえば 1d2 です。

●NAN、0x12、INF、INFINITY などの特殊な値。

●NULL (例：SELECT ISNUMERIC(NULL);)

標準と互換性

● **SQL/2008** ベンダー拡張。

例

次の例は、外部ファイルからデータをインポートし、無効な値があるローをエクスポートし、残りのローを永久テーブルにコピーします。この例では、ISNUMERIC 文を使用して height_in_cms 値が数値であることを検証します。

```
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;
```

LAST_VALUE 関数 [集合]

ウィンドウの最後のローの値を返します。

構文

LAST_VALUE([**ALL**] *expression*[{ **RESPECT** | **IGNORE** } **NULLS**] **)**
OVER (*window-spec* **)**

window-spec :次の「備考」の項を参照してください

パラメーター

● ***expression*** 評価する式。たとえば、カラム名です。

戻り値

引数のデータ型。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

LAST_VALUE 関数を使用すると、セルフジョインを使用せずに、(何らかの順序による) 最後の値を選択できます。最後の値を計算の基準として使用する場合、この関数が役立ちます。

LAST_VALUE 関数は、ORDER BY を実行した後の分割から最後のレコードを取得します。次に、最後のレコードに対して *expression* が比較され、結果が返されます。

IGNORE NULLS を指定すると、*expression* にある最後の NULL 以外の値が返されます。RESPECT NULLS (デフォルト) を指定すると、最後の値が、それが NULL であってもなくても返されます。

LAST_VALUE 関数は、その他の大部分の集合関数とは異なり、ウィンドウ指定を行った場合にのみ使用できます。

window-spec の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。「WINDOW 句」989 ページを参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「Window 集合関数」『SQL Anywhere サーバー SQL の使用法』
- 「FIRST_VALUE 関数 [集合]」230 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

  SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」がサポートされています。これにより、カラム参照ではない任意の式を Window 関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、LAST_VALUE 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例は、各従業員の給料と、同じ部署内で最高額の給料を受け取っている従業員の名前を返します。

```
SELECT GivenName + ' ' + Surname AS employee_name,
    Salary, DepartmentID,
    LAST_VALUE( employee_name ) OVER Salary_Window AS highest_paid
FROM 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
...	...	...	...

部署 500 で最高額の給料を受け取っている従業員は Jose Martinez で、部署 400 で最高額の給料を受け取っている従業員は Scott Evans です。

LCASE 関数 [文字列]

文字列中のすべての文字を小文字に変換します。

構文

LCASE(*string-expression* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***string-expression*** 小文字に変換される文字列。

戻り値

- CHAR
- NCHAR
- LONG VARCHAR
- VARCHAR
- NVARCHAR

備考

LCASE 関数は LOWER 関数と同じです。

参照

- 「LOWER 関数 [文字列]」273 ページ
- 「UCASE 関数 [文字列]」378 ページ
- 「UPPER 関数 [文字列]」381 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。同義の関数 LOWER は、SQL/2008 標準のコア機能です。

例

次の文は、値 chocolate を返します。

```
SELECT LCASE( 'ChoCOlatE' );
```

LEFT 関数 [文字列]

文字列の先頭からいくつかの文字を返します。

構文

LEFT(*string-expression*, *integer-expression* **)**

パラメーター

- ***string-expression*** 文字列。
- ***integer-expression*** 返す文字数。

戻り値

- LONG VARCHAR
- LONG NVARCHAR

備考

文字列にマルチバイト文字があり、適切な照合が使用されている場合、返されるバイト数が指定した文字数よりも大きい場合があります。

引数 string-expression の値より大きな *integer-expression* を指定できます。この場合、全体の値が返されます。

この関数は NCHAR の入力または出力をサポートしています。入力文字が文字長のセマンティックを使用している場合、可能であれば、戻り値が文字長のセマンティックで記述されます。

参照

- 「RIGHT 関数 [文字列]」334 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、Customers テーブルに含まれる各 Surname 値の最初の 5 文字を返します。

```
SELECT LEFT( Surname, 5) FROM Customers;
```

LENGTH 関数 [文字列]

指定した文字列の文字数を返します。

構文

{ LENGTH | LEN }(*string-expression* **)**

パラメーター

- ***string-expression***　文字列。

戻り値

INT

備考

この関数を使用すると、文字列の長さがわかります。たとえば、*string-expression* にカラム名を指定すると、カラムの値の長さがわかります。

文字列にマルチバイト文字があり、適切な照合が使用されている場合、LENGTH はバイト数ではなく、文字数を返します。文字列が BINARY データ型の場合、LENGTH 関数は BYTE_LENGTH 関数のように動作します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **注意**
> データ型が CHAR、VARCHAR、LONG VARCHAR、NCHAR の場合、LENGTH 関数と CHAR_LENGTH 関数を使用すると同じ結果が得られます。ただし、BINARY データ型とビット配列データ型には LENGTH 関数を使用します。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「BYTE_LENGTH 関数 [文字列]」164 ページ
- 「国際言語と文字セット」『SQL Anywhere サーバー データベース管理』
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　LENGTH 関数はベンダー拡張ですが、セマンティックは SQL/2008 標準の CHAR_LENGTH 関数と同じです。 NCHAR データ型の文字列式での LENGTH の使用は、オプションの SQL/2008 言語機能 F421 の一部です。

例

次の文は、値 9 を返します。

```
SELECT LENGTH( 'chocolate' );
```

LESSER 関数 [その他]

2 つのパラメーター値のうち、より小さい値を返します。

構文

LESSER(*expression-1*, *expression-2* **)**

パラメーター

- ***expression-1***　比較される最初のパラメーター値。
- ***expression-2***　比較される 2 番目のパラメーター値。

戻り値

この関数の戻り値は、指定した式によって異なります。具体的には、データベースサーバーが関数を評価するとき、まず、式の比較が可能なデータ型を検索します。該当するデータ型が見つかると、データベースサーバーは式を比較し、比較に使用したデータ型で結果を返します。データベースサーバーは、一般に比較が可能なデータ型を見つけることができないと、エラーを返します。

備考

パラメーターが等しい場合は、最初の値を返します。

参照

- 「GREATER 関数 [その他]」238 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 5 を返します。

```
SELECT LESSER( 10, 5 ) FROM dummy;
```

LIST 関数 [集合]

グループ内のローごとに値のデリミター付きリストを返します。

構文

LIST(
[**ALL** | **DISTINCT**] *string-expression*
[, *delimiter-string*]
[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...] **)**

パラメーター

- ***string-expression***　文字列式。通常はカラム名です。ALL が指定されていると (デフォルト)、グループ内のローごとに *string-expression* の値が結果文字列に追加されます。各値は、*delimiter-string* で区切られます。DISTINCT が指定されていると、ユニークな *string-expression* 値のみが追加されます。

- ***delimiter-string***　リスト項目のデリミター文字列。デフォルト設定はカンマです。NULL 値または空の文字列を指定した場合は、デリミターはありません。*delimiter-string* は定数です。

- ***order-by-expression***　関数によって返された項目を並べ替えます。この引数の前にカンマは必要ありません。このため、*delimiter-string* を指定しない場合、使用が簡単になります。

  order-by-expression には整数リテラルを指定できません。ただし、整数リテラルを含む変数を指定できます。

  ORDER BY 句に定数が含まれている場合、それらの定数はオプティマイザーによって解釈され、同義の ORDER BY 句に置き換えられます。たとえば、オプティマイザーは ORDER BY 'a' を ORDER BY 式として解釈します。

  クエリブロックに、有効な ORDER BY 句が指定された複数の集合関数が含まれているとき、それらの ORDER BY 句を単一の ORDER BY 句に論理的に結合できる場合は、そのクエリブロックを実行できます。たとえば、次の ORDER BY 句の場合は、

  ```
  ORDER BY expression1, 'a', expression2
  ORDER BY expression1, 'b', expression2, 'c', expression3
  ```

  次の ORDER BY 句として結合されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
ORDER BY expression1, expression2, expression3
```

戻り値

- LONG VARCHAR
- LONG NVARCHAR

備考

LIST 関数は、グループの各ローの、NULL 以外のすべての X の値を (デリミター付きで) 連結して返します。グループ内に明確な X 値を持ったローが 1 つ以上存在しない場合、LIST(X) は空の文字列を返します。

LIST 関数では、NULL 値と空の文字列は無視されます。

LIST 関数は Window 関数として使用できませんが、Window 関数の入力には使用できます。

この関数は NCHAR の入力または出力をサポートしています。

標準と互換性

- **SQL/2008** ベンダー拡張。

  SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」がサポートされています。これにより、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、LIST 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。「集合関数と外部参照」『SQL Anywhere サーバー SQL の使用法』を参照してください。

  例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「sa_split_list システムプロシージャー」1194 ページ

例

次の文は、値 487 Kennedy Court, 547 School Street を返します。

```
SELECT LIST( Street ) FROM Employees
WHERE GivenName = 'Thomas';
```

次の文は、従業員 ID をリストします。結果セットの各ローには、部門ごとの従業員 ID のカンマで区切られたリストが入っています。

```
SELECT LIST( EmployeeID )
FROM 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,...

次の文は、従業員の姓を基準に従業員 ID をソートします。

```
SELECT LIST( EmployeeID ORDER BY Surname ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

Sorted IDs '1751,591,1062,1191,992,888,318,184,1576,207,1684,1643,1607,1740,409,1507'

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,...
160,105,1250,247,266,249,445,...

次の文は、セミコロンで区切られたリストを返します。ORDER BY 句とリストセパレーターの位置に注意してください。

```
SELECT LIST( EmployeeID, ';' ORDER BY Surname ) AS "Sorted IDs"
FROM 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;...

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

前の文と次の文は区別してください。次の文では、(Surname, ';') を複合ソートキーに使用してソートした従業員 ID のカンマ区切りのリストが返ります。

```
SELECT LIST( EmployeeID ORDER BY Surname, ';' ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

LOCATE 関数 [文字列]

異なる文字列内のある文字列の位置を返します。

構文

LOCATE(*string-expression-1*, *string-expression-2* [, *integer-expression*] **)**

パラメーター

- ***string-expression-1*** 検索される文字列。
- ***string-expression-2*** 検索する文字列。この文字列は 255 バイトに制限されています。
- ***integer-expression*** 文字列内の、検索を開始する位置。最初の文字の位置は 1 です。開始オフセットが負の場合は、最初に一致した文字列ではなく、最後に一致した文字列のオフセットを返します。負のオフセットは、文字列の末尾のどれだけの部分が検索から除外されるかを示します。除外されるバイト数は、(-1 * offset) -1 で計算されます。

戻り値

INT

備考

integer-expression を指定すると、文字列のオフセットから検索が開始します。

最初の文字列には長い文字列 (255 バイト以上) を指定できますが、2 番目の文字列は 255 バイトに制限されます。長い文字列を 2 番目の引数として指定すると、LOCATE 関数は NULL 値を返します。文字列が見つからない場合は、0 を返します。長さ 0 の文字列を検索すると、1 を返します。いずれかの引数が NULL の場合は、結果は NULL です。

マルチバイト文字が使用され、適切な照合が使用されている場合は、開始位置と返される値は**バイト**で計算した位置と異なる場合があります。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「文字列関数」148 ページ
- 「CHARINDEX 関数 [文字列]」169 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 8 を返します。

```
SELECT LOCATE(
  'office party this week - rsvp as soon as possible',
  'party',
  2 );
```

次の文は、

```
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;
```

次の出力を返します。

str	pos	path	filename
c:¥test¥functions ¥locate.sql	18	c:¥test¥functions	locate.sql

LOG 関数 [数値]

数の自然対数を返します。

構文

LOG(*numeric-expression* **)**

パラメーター

- ***numeric-expression*** 数値。

戻り値

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメーターが NULL 値の場合、結果は NULL 値になります。

備考

引数は、組み込みの数値データ型の値を返す式です。

参照

- 「LOG10 関数 [数値]」273 ページ

標準と互換性

- **SQL/2008**　SQL/2008 標準では、キーワード LN を使用して自然対数関数が定義されます。自然対数関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の文は、50 の自然対数を返します。

```
SELECT LOG( 50 );
```

LOG10 関数 [数値]

数値の対数 (基数 10) を返します。

構文

LOG10(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　数値。

戻り値

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。パラメーターが NULL 値の場合、結果は NULL 値になります。

備考

引数は、組み込みの数値データ型の値を返す式です。

参照

- 「LOG 関数 [数値]」272 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、10 を底とする 50 の対数を返します。

```
SELECT LOG10( 50 );
```

LOWER 関数 [文字列]

文字列中のすべての文字を小文字に変換します。この関数は LCASE 関数と同じです。

構文

LOWER(*string-expression* **)**

パラメーター

- ***string-expression*** 小文字に変換される文字列。

戻り値

引数のデータ型に対応する CHAR、VARCHAR、LONG VARCHAR、NCHAR、NVARCHAR、LONG NVARCHAR。

備考

LCASE 関数は LOWER 関数と同じです。

参照

- 「LCASE 関数 [文字列]」264 ページ
- 「UCASE 関数 [文字列]」378 ページ
- 「UPPER 関数 [文字列]」381 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** LOWER 関数は、SQL/2008 標準のコア機能です。 NCHAR データ型の式での LOWER の使用は、オプションの SQL/2008 言語機能 F421 の一部です。

例

次の文は、値 chocolate を返します。

```
SELECT LOWER( 'chOCOLate' );
```

LTRIM 関数 [文字列]

文字列の先行ブランクを削除します。

構文

LTRIM(*string-expression* **)**

パラメーター

- ***string-expression*** 削除する文字列。

戻り値

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

備考

結果の実際の長さは、式の長さから、削除する文字数を引いた値です。すべての文字を削除すると、結果は空の文字列になります。

パラメーターに NULL を指定できる場合、結果は NULL になります。

パラメーターが NULL の場合、結果は NULL 値になります。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「RTRIM 関数 [文字列]」338 ページ
- 「TRIM 関数 [文字列]」375 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

  SQL/2008 標準で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数で提供されます。

例

次の文は、すべての先行ブランクを削除して、値 Test Message を返します。

```
SELECT LTRIM( '    Test Message' );
```

MAX 関数 [集合]

各ローグループで見つかった *expression* の最大値を返します。

構文 1

MAX([ALL | DISTINCT] *expression* **)**

構文 2

MAX([ALL] *expression* **) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- **[ALL] *expression***　最大値が計算される式。通常はカラム名です。
- **DISTINCT *expression***　MAX(*expression*) の場合と同じ値を返します。万全を期すために含まれています。

戻り値

引数と同じデータ型。

備考

expression が NULL のローは、無視されます。グループにローが含まれていない場合は、NULL を返します。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。「WINDOW 句」989 ページを参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

この関数は NCHAR の入力または出力をサポートしています。

また、2 つの式を単純に比較する場合は、GREATER 関数を使用することもできます。

参照

- 「GREATER 関数 [その他]」238 ページ
- 「MIN 関数 [集合]」278 ページ

標準と互換性

- **SQL/2008**　コア機能。Window 関数として使用する場合 (構文 2)、MAX はオプションの SQL/2008 言語機能 T611、「Elementary OLAP operations」の一部です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、MAX 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。

  例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「集合関数と外部参照」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、Employees テーブル内の給与の最大値 138948.000 を返します。

```
SELECT MAX( Salary )
FROM Employees;
```

MEDIAN 関数 [集合]

ローのセットの数値式の中央値を計算します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 1

MEDIAN([ALL | DISTINCT] *numeric-expression* **)**

構文 2

MEDIAN([ALL] *numeric-expression* **) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***numeric-expression*** ローセットで中央値が計算される式。
- **DISTINCT 句** 入力に含まれるユニークな値の中央値を計算する前に、重複した値を除外します。
- **ALL 句** 入力に含まれるすべての値 (重複した値も含む) の中央値を計算します。これはデフォルトの動作です。

戻り値

戻り値のデータ型は、入力値と同じです。

中央値の計算では、NULL は無視されます。ただし、ローが含まれないグループについては NULL 値が返されます。

備考

numeric-expression 値には、BIT を除くどの数値データ型でも使用できます。「数値データ型」93 ページを参照してください。

数値の有限リストの中央値は、リスト内の値を最小値から最大値まで順に並べ、その中央にある値を選択することで特定できます。数値が偶数存在する場合は、中央値がユニークにならないため、2 つの中央値の平均が MEDIAN 関数から返されます。多くとも、母集団の半分の要素が中央値より小さく、残りの半分が中央値より大きくなります。両方のグループの要素の数がそれぞれ母集団の半分未満の場合は、中央値と完全に等しい値を持つ要素が存在します。たとえば、a < b < c の場合、リスト {a, b, c} の中央値は b になります。a < b < c < d の場合は、リスト {a, b, c, d} の中央値は b と c の平均である ((b + c)/2) になります。

中間の 2 つの要素を平均した結果に小数点以下の数値が含まれ、かつ入力データ型で小数部分を表すことができない場合は、小数点以下の数値がトランケートされます。このトランケートを回避するには、小数点以下桁数が有効な数値型に入力をキャストします。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。「WINDOW 句」989 ページの *window-spec* 定義を参照してください。

window-spec にはパーティションのみを指定できます (ROW または RANGE の指定を含めることはできません)。WINDOW 句を使用した場合、DISTINCT はサポートされません。CUBE、ROLLUP、および GROUPING SETS は、構文 1 でサポートされます。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「SUM 関数 [集合]」363 ページ
- 「COUNT 関数 [集合]」183 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。Window 関数は、オプションの SQL/2008 言語機能 T611、「Elementary OLAP operations」の一部です。

  SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」がサポートされています。これにより、カラム参照ではない任意の式を Window 関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、MEDIAN 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の文は、Employees テーブルから給与の中央値を返します。

```
SELECT MEDIAN( Salary ) FROM Employees;
```

次の文は、Employees テーブルから州ごとの給与の中央値を返します。

```
SELECT EmployeeID, Surname, Salary, State,
  MEDIAN( Salary ) OVER Salary_Window
FROM Employees
WINDOW Salary_Window AS ( PARTITION BY State )
ORDER BY State, Surname;
```

MIN 関数 [集合]

各ローグループで見つかった expression の最小値を返します。

構文 1

MIN([ALL | DISTINCT] *expression* **)**

構文 2

MIN([ALL] *expression* **) OVER (** *window-spec* **)**

window-spec :以下の備考部分の構文 2 の説明を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **[ALL] *expression*** 最小値が計算される式。通常はカラム名です。

- **DISTINCT *expression*** MIN(*expression*) の場合と同じ値を返します。万全を期すために含まれています。

戻り値

引数と同じデータ型。

備考

expression が NULL のローは、無視されます。グループにローが含まれていない場合は、NULL を返します。

この関数は NCHAR の入力または出力をサポートしています。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。詳細については、WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

また、2 つの式を単純に比較する場合は、LESSER 関数を使用することもできます。「LESSER 関数 [その他]」267 ページを参照してください。

参照

- 「MAX 関数 [集合]」275 ページ
- 「WINDOW 句」989 ページ

標準と互換性

- **SQL/2008** コア機能。Window 関数として使用する場合 (構文 2)、MIN はオプションの SQL/2008 言語機能 T611、「Elementary OLAP operations」の一部です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、MIN 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。「集合関数と外部参照」『SQL Anywhere サーバー SQL の使用法』を参照してください。

例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の文は、Employees テーブル内の給与の最小値 24903.000 を返します。

```
SELECT MIN( Salary )
FROM Employees;
```

MINUTE 関数 [日付と時刻]

TIMESTAMP 値の分部分を返します。

構文

MINUTE(*timestamp-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP 値。

戻り値

SMALLINT

備考

TIMESTAMP 式の分部分である、0 ～ 59 の間の SMALLINT 値が返されます。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 22 を返します。

```
SELECT MINUTE( '1998-07-13 12:22:34' );
```

MINUTES 関数 [日付と時刻]

MINUTES 関数は、TIMESTAMP を操作するか、2 つの TIMESTAMP 値の間の分数を返します。次の「備考」の項を参照してください。

構文 1

MINUTES(*timestamp-expression* **)**

構文 2

MINUTES(*timestamp-expression*, *timestamp-expression* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 3

MINUTES(*timestamp-or-time-expression*, *integer-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP データ型の式。
- ***timestamp-or-time-expression*** TIME または TIMESTAMP データ型の式。
- ***integer-expression*** *timestamp-or-time-expression* に加算される分数。*integer-expression* が負の場合、*timestamp-or-time-expression* から適切な分数が引かれます。

戻り値

構文 1 または構文 2 では INTEGER。

構文 3 では TIME または TIMESTAMP。

備考

MINUTES 関数の結果は、その引数によって異なります。

- **構文 1** 1 つの *timestamp-expression* を MINUTES 関数に渡すと、0000-02-29 の真夜中から *timestamp-expression* までの分数を INTEGER として返します。

  注意
  0000-02-29 は実際の日付を指すための値ではありません。MINUTES 関数で使用されるデフォルトの日付です。

- **構文 2** 2 つの TIMESTAMP 値を MINUTES 関数に渡すと、その間の分数を整数で返します。
- **構文 3** TIMESTAMP 値と INTEGER 値を MINUTES 関数に渡すと、整数の分数を *timestamp-expression* 引数に加算した TIMESTAMP 結果を返します。同様に、MINUTES の最初の引数が TIME 値の場合は、結果も TIME 値になります。構文 3 では、最初の引数の暗黙的変換をサポートしていません。最初の引数を DATE、TIME または TIMESTAMP の値に明示的にキャストしてください。最初の引数が DATE データ型の場合、時刻部分には真夜中が使用されます。

MINUTES は整数を返すので、構文 1 で 4083-03-23 02:08:00 以上の TIMESTAMP 値を使用すると、オーバーフローが起こる場合があります。

構文 2 の代わりに、DATEDIFF 関数を使用します。構文 3 の代わりに、DATEADD 関数を使用します。

参照

- 「DATEDIFF 関数 [日付と時刻]」196 ページ
- 「DATEADD 関数 [日付と時刻]」194 ページ
- 「CAST 関数 [データ型変換]」165 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 240 を返します。これは、2 番目の TIMESTAMP 値が、最初のタイムスタンプの 240 分後であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。

```
SELECT MINUTES( '1999-07-13 06:07:12',
   '1999-07-13 10:07:12' );

SELECT DATEDIFF( minute,
    '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' );
```

次の文は、値 1051040527 を返します。

```
SELECT MINUTES( '1998-07-13 06:07:12' );
```

次の文は、TIMESTAMP 値 1999-05-12 21:10:07.000 を返します。最初の文では、リテラル文字列パラメーターの明示的なキャストが必要であることに注意してください。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5);

SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```

次の文は 'TIME' を返します。TIME 引数を指定して MINUTES 関数が呼び出された場合に、TIME 値を返すことを示します。

```
SELECT EXPRTYPE('SELECT MINUTES( CAST( "13:45:00.000" AS TIME ), 16 )', 1);
```

MOD 関数 [数値]

整数を整数で割ったときの余りを返します。

構文

MOD(*dividend*, *divisor* **)**

パラメーター

- ***dividend***　被除数 (分数の分子)。
- ***divisor***　除数 (分数の分母)。

戻り値

- SMALLINT
- INT
- NUMERIC

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

被除数が負の場合、除算の結果は負または 0 になります。除数の符号は計算結果に影響を与えません。

参照

- 「REMAINDER 関数 [数値]」328 ページ

標準と互換性

- **SQL/2008**　MOD 関数は、オプションの SQL/2008 言語機能 T441 の一部です。

例

次の文は、値 2 を返します。

```
SELECT MOD( 5, 3 );
```

MONTH 関数 [日付と時刻]

指定した日付の月を返します。

構文

MONTH(*date-expression* **)**

パラメーター

- ***date-expression***　DATE データ型の値。

戻り値

SMALLINT

備考

戻り値は、指定の日付の月に対応する 1 ～ 12 の間の数です。

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 7 を返します。

```
SELECT MONTH( '1998-07-13' );
```

MONTHNAME 関数 [日付と時刻]

日付から月の名前を返します。

構文

MONTHNAME(*date-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP 値。

戻り値

VARCHAR

備考

結果が数値 (2 月を示す 2 など) の場合でも、MONTHNAME 関数は文字列を返します。

参照

- 「DATEPART 関数 [日付と時刻]」 199 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 September を返します。

```
SELECT MONTHNAME( '1998-09-05' );
```

MONTHS 関数 [日付と時刻]

MONTHS 関数は、TIMESTAMP を操作するか、2 つの TIMESTAMP 値の間の月数を返します。次の「備考」の項を参照してください。

構文 1

MONTHS(*timestamp-expression* **)**

構文 2

MONTHS(*timestamp-expression*, *timestamp-expression* **)**

構文 3

MONTHS(*timestamp-expression*, *integer-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP データ型の日付と時刻。

- ***integer-expression*** *timestamp-expression* に加算される整数の月数 (SMALLINT データ型)。*integer-expression* が負の場合、*timestamp-expression* から適切な月数が引かれます。*integer-expression* を指定する場合は、*timestamp-expression* を TIME、DATE または TIMESTAMP データ型として明示的にキャストしてください。*timestamp-expression* が TIME 値の場合、現在の月が使用されます。

データ型のキャストの詳細については、「CAST 関数 [データ型変換]」165 ページを参照してください。

戻り値

構文 1 または構文 2 では INTEGER。

構文 3 では TIMESTAMP。

備考

MONTHS 関数の結果は、その引数によって異なります。MONTHS 関数では、引数内の時間、分、および秒は無視されます。

- **構文 1**　1 つの *timestamp-expression* を MONTHS 関数に渡すと、0000-02 から *timestamp-expression* までの月数を INTEGER として返します。

> **注意**
> 0000-02-29 は実際の日付を指すための値ではありません。MONTHS 関数で使用されるデフォルトの日付です。

- **構文 2**　2 つの TIMESTAMP 値を MONTHS 関数に渡すと、その間の月数を整数で返します。
- **構文 3**　TIMESTAMP 値と SMALLINT 値を MONTHS 関数に渡すと、整数の月数を *timestamp-expression* に加算した TIMESTAMP 結果を返します。

構文 2 の代わりに、DATEDIFF 関数を使用します。構文 3 の代わりに、DATEADD 関数を使用します。

MONTHS の値を求めるには、2 つの日付の間に月の最初の日がいくつあるかを計算します。

参照

- 「DATEDIFF 関数 [日付と時刻]」196 ページ
- 「DATEADD 関数 [日付と時刻]」194 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 2 を返します。これは、2 番目の日付が、最初の日付の 2 か月後であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。

```
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' );
```

次の文は、値 23981 を返します。

```
SELECT MONTHS( '1998-07-13 06:07:12' );
```

次の文は、TIMESTAMP 値 1999-10-12 21:05:07.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5);
SELECT DATEADD( month, 5, '1999-05-12 21:05:07' );
```

NCHAR 関数 [文字列]

Unicode のコードポイントがパラメーターに指定された 1 文字を含む NCHAR 文字列を返します。値が有効なコードポイント値ではない場合は、NULL を返します。

構文

NCHAR(*integer* **)**

パラメーター

- ***integer*** 対応する Unicode コードポイントに変換される数値。

戻り値

NVARCHAR

参照

- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」175 ページ
- 「TO_NCHAR 関数 [文字列]」370 ページ
- 「TO_CHAR 関数 [文字列]」369 ページ
- 「UNICODE 関数 [文字列]」379 ページ
- 「UNISTR 関数 [文字列]」380 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、アラビア文字の ALEF (Unicode コードポイント U+627) を返します。

```
SELECT NCHAR( 1575 );
```

NEWID 関数 [その他]

UUID (ユニバーサルユニーク識別子) 値を生成します。UUID は、GUID (グローバルユニーク識別子) と同じです。

構文

NEWID( )

パラメーター

NEWID 関数に関連付けられているパラメーターはありません。

戻り値

UNIQUEIDENTIFIER

備考

NEWID 関数は、カラムの DEFAULT 句で使用できます。

UUID を使用して、テーブルのローをユニークに識別できます。コンピューターが異なると生成される値も異なるので、値は同期環境やレプリケーション環境でキーとして使用できます。

他の RDBMS と互換性を保つために、UUID にはハイフンが含まれます。このデフォルト設定は、uuid_has_hyphens option を Off にして変更することができます。詳細については、「uuid_has_hyphens オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

NEWID 関数は、非決定的関数です。以降、NEWID 関数を呼び出すと異なる値が返されます。クエリオプティマイザーは、NEWID 関数の結果をキャッシュしません。

非決定的関数の詳細については、「関数のキャッシュ」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「NEWID デフォルト」『SQL Anywhere サーバー SQL の使用法』
- 「STRTOUUID 関数 [文字列]」359 ページ
- 「UUIDTOSTR 関数 [文字列]」383 ページ
- 「UNIQUEIDENTIFIER データ型」123 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、2 つのカラムを持つテーブル mytab を作成します。カラム pk は uniqueidentifier データ型とし、NEWID 関数をデフォルト値として割り当てます。カラム c1 は integer データ型です。

```
CREATE TABLE mytab(
  pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
  c1 INT );
```

次の文は、ユニーク識別子を文字列として返します。

```
SELECT UUIDTOSTR( NEWID() );
```

たとえば、戻り値が 96603324-6FF6-49DE-BF7D-F44C1C7E6856 の場合もあります。

NEXT_CONNECTION 関数 [システム]

次の接続の識別番号を返します。

構文

NEXT_CONNECTION(*connection-id* [, *database-id*] **)**

戻り値

INT

パラメーター

- ***connection-id*** 整数。通常は、前回の NEXT_CONNECTION の呼び出しから返された整数です。*connection-id* が NULL の場合、NEXT_CONNECTION は最新の接続 ID を返します
- ***database-id*** 現在のサーバー上のいずれかのデータベースを表す整数。*database-id* を指定しない場合は、現在のデータベースが使用されます。NULL を指定した場合、NEXT_CONNECTION はデータベースに関係なく、次の接続を返します。

備考

NEXT_CONNECTION を使用して、データベースへの接続を列挙できます。通常、接続 ID は単調に増加する昇順で作成されます。この関数は次の接続 ID を降順で返します。

最新の接続の接続 ID 値を取得するには、*connection-id* に NULL を入力します。以降の接続を取得するには、その前の戻り値を入力します。その順序で接続がなくなると、NULL を返します。

NEXT_CONNECTION が便利なのは、指定した時点以前に作成されたすべての接続を解除するときです。ただし、NEXT_CONNECTION は接続 ID を降順で返すため、関数の開始後に作成された接続は返されません。すべての接続を確実に切断するには、NEXT_CONNECTION を実行する前に新しい接続を作成しないようにします。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、現在のデータベースでの最初の接続に使用する識別子を返します。識別子は、10 のような整数値です。

```
SELECT NEXT_CONNECTION( NULL );
```

次の文は、5 のような値を返します。

```
SELECT NEXT_CONNECTION( 10 );
```

次の呼び出しは、現在のデータベースで指定した *connection-id* から降順で次の接続 ID を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT NEXT_CONNECTION( connection-id );
```

次の呼び出しは、データベースに関係なく指定した *connection-id* から降順で次の接続 ID を返します。

```
SELECT NEXT_CONNECTION( connection-id, NULL );
```

次の呼び出しは、指定したデータベースで指定した *connection-id* から降順で次の接続 ID を返します。

```
SELECT NEXT_CONNECTION( connection-id, database-id );
```

次の呼び出しは、データベースに関係なく最初の接続を返します。

```
SELECT NEXT_CONNECTION( NULL, NULL );
```

次の呼び出しは、指定されたデータベースの最初の接続を返します。

```
SELECT NEXT_CONNECTION( NULL, database-id );
```

NEXT_DATABASE 関数 [システム]

データベースの識別番号を返します。

構文

NEXT_DATABASE(*database-id* **)**

パラメーター

- ***database-id*** データベースの ID 番号を指定する整数。

戻り値

INT

備考

NEXT_DATABASE 関数を使用して、データベースサーバーで実行中のデータベースを列挙できます。最初のデータベースを取得するには、NULL を渡します。その後の各データベースを取得するには、前回の戻り値を渡します。データベースがなくなると、NULL を返します。データベース ID 番号は特定の順序で返されるわけではありませんが、データベース ID を使用して、サーバーで開始されたデータベースの順序を知ることができます。サーバーで最初に開始されたデータベースには値 0 が割り当てられ、サーバーでその後に開始されたデータベースには、データベース ID が 1 ずつ増分されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで呼び出された場合、現在のテナントデータベースについての情報のみを返します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、最初のデータベースの値である 0 を返します。

```
SELECT NEXT_DATABASE( NULL );
```

次の文は、1 つのデータベースが起動している場合のみ NULL を返します。

```
SELECT NEXT_DATABASE( 0 );
```

参照

- 「DB_NAME 関数 [システム]」208 ページ
- 「sa_db_list システムプロシージャー」1067 ページ

NEXT_HTTP_HEADER 関数 [Web サービス]

次の HTTP ヘッダーの名前を返します。

構文

NEXT_HTTP_HEADER(*header-name* **)**

パラメーター

- ***header-name*** 前の要求ヘッダーの名前。header-name が NULL の場合、この関数は最初の HTTP 要求ヘッダーの名前を返します。

戻り値

LONG VARCHAR

備考

この関数を HTTP 要求ヘッダーで繰り返し使用することで、次の HTTP ヘッダー名が返されます。NULL を指定して呼び出すと、最初のヘッダーの名前が返されます。後続のヘッダーは、前のヘッダーの名前を関数に渡すことによって取得されます。この関数を最後のヘッダー名を使用して呼び出した場合、または Web サービスから呼び出していない場合、NULL を返します。

この関数を繰り返し呼び出すと、すべてのヘッダーフィールドが一度だけ返されます。ただし、必ずしも HTTP 要求での表示順に表示されるとはかぎりません。

参照

- 「HTTP_HEADER 関数 [Web サービス]」249 ページ
- 「sa_http_header_info システムプロシージャー」1097 ページ
- 「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は、HTTP 要求ヘッダーの名前と値をデータベースサーバーメッセージウィンドウに表示します。

```
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;
```

NEXT_HTTP_RESPONSE_HEADER 関数 [Web サービス]

次の HTTP 応答ヘッダーの名前を取得します。

構文

NEXT_HTTP_RESPONSE_HEADER(*header-name* **)**

パラメーター

- ***header-name*** 前の応答ヘッダーの名前。header-name が NULL の場合、この関数は最初の HTTP 応答ヘッダーの名前を返します。

戻り値

LONG VARCHAR

備考

この関数を HTTP 応答ヘッダーで繰り返し使用することで、次の HTTP 応答ヘッダー名が返されます。NULL を指定して呼び出すと、最初の応答ヘッダーの名前が返されます。後続の応答ヘッダーは、前の応答ヘッダーの名前を関数に渡すことによって取得されます。この関数は、最後の応答ヘッダー名を使用して呼び出した場合、または Web サービスから呼び出していない場合、NULL を返します。

この関数を繰り返し呼び出すと、すべての応答ヘッダーフィールドが 1 回のみ返されます。ただし、必ずしも HTTP 応答での表示順に表示されるとはかぎりません。

参照

- 「HTTP_RESPONSE_HEADER 関数 [Web サービス]」251 ページ
- 「HTTP 要求ヘッダーの管理」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は、HTTP 応答ヘッダーの名前と値をデータベースサーバーメッセージウィンドウに表示します。

```
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;
```

NEXT_HTTP_VARIABLE 関数 [Web サービス]

次の HTTP 変数の名前を返します。

構文

NEXT_HTTP_VARIABLE(*var-name* **)**

パラメーター

- ***var-name***　前の変数の名前。*var-name* が NULL の場合、この関数は最初の HTTP 変数の名前を返します。

戻り値

LONG VARCHAR

備考

この関数は、要求内の HTTP 変数に対して反復して適用されます。NULL を指定して呼び出すと、最初の変数の名前が返されます。後続の変数は、前の変数の名前を関数に渡すことによって取得されます。この関数は、最後の変数名を使用して呼び出した場合、または Web サービスから呼び出していない場合、NULL を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

この関数を繰り返し呼び出すと、すべての変数が一度だけ返されます。ただし、必ずしも HTTP 要求での表示順に表示されるとはかぎりません。URL PATH が ON または ELEMENTS に設定される場合、変数 url または url1、url2、...、url10 が個々に含められます。

参照

- 「HTTP_VARIABLE 関数 [Web サービス]」253 ページ
- 「NEXT_HTTP_HEADER 関数 [Web サービス]」290 ページ
- 「sa_http_variable_info システムプロシージャー」1100 ページ
- 「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は最初の HTTP 変数の名前を返します。

```
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;
```

NEXT_SOAP_HEADER 関数 [SOAP]

SOAP 要求ヘッダーの次のヘッダーキーを返します。

構文

NEXT_SOAP_HEADER(*header-key* **)**

パラメーター

- ***header-key***　指定したヘッダーエントリの最上位 XML 要素の XML ローカル名。

戻り値

LONG VARCHAR

備考

header-key に NULL を指定すると、この関数は SOAP ヘッダーで見つかった最初のヘッダーエントリのヘッダーキーが返されます。

最後の *header-key* を使用して呼び出すと、この関数は NULL が返されます。

参照

- 「SOAP_HEADER 関数 [SOAP]」346 ページ
- 「チュートリアル：SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は SOAP 要求ヘッダーにあるすべてのキーを処理します。**Authentication** キーを処理するときに、キーの値も取得します。

```
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;
```

NOW 関数 [日付と時刻]

現在の日付と時刻を TIMESTAMP 値で返します。精度はシステムクロックの精度によって制限されます。

構文

NOW([*])

戻り値

TIMESTAMP

備考

NOW は、GETDATE 関数および CURRENT TIMESTAMP 特別値と同義です。NOW(*) と NOW() の構成は同義です。

要求に含まれる NOW 関数の各インスタンスは、多くても 1 回のみ評価されます。同じ要求に含まれる NOW の複数のインスタンスは、同じ TIMESTAMP 値を共有する場合もあれば、そうでない場合もあります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CURRENT TIME 特別値」63 ページ
- 「CURRENT TIMESTAMP 特別値」64 ページ
- 「CURRENT UTC TIMESTAMP 特別値」66 ページ
- 「DATE データ型」112 ページ
- 「DATE 関数 [日付と時刻]」193 ページ
- 「DATETIME データ型」113 ページ
- 「DATETIME 関数 [日付と時刻]」200 ページ
- 「DATETIMEOFFSET データ型」114 ページ
- 「式」12 ページ
- 「GETDATE 関数 [日付と時刻]」235 ページ
- 「ISDATE 関数 [データ型変換]」258 ページ
- 「SMALLDATETIME データ型」116 ページ
- 「TIME データ型」117 ページ
- 「TIMESTAMP 特別値」73 ページ
- 「TIMESTAMP データ型」118 ページ
- 「UTC TIMESTAMP 特別値」75 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、現在の日付と時刻を返します。

```
SELECT NOW( * );
```

NULLIF 関数 [その他]

式を比較して、CASE 式の省略形を提供します。

構文

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

パラメーター

- ***expression-1***　比較される式。
- ***expression-2***　比較される式。

戻り値

最初の引数のデータ型。

備考

NULLIF は 2 つの式の値を比較します。

1 番目の式と 2 番目の式が等しい場合、NULLIF は NULL を返します。

1 番目の式と 2 番目の式が異なる場合、または 2 番目の式が NULL の場合、NULLIF は 1 番目の式を返します。

NULLIF 関数は、いくつかの CASE 式の簡単な作成方法を提供します。

参照

- 「CASE 式」16 ページ

標準と互換性

- **SQL/2008** コア機能。

例

次の文は、値 a を返します。

```
SELECT NULLIF( 'a', 'b' );
```

次の文は、NULL を返します。

```
SELECT NULLIF( 'a', 'a' );
```

NUMBER 関数 [その他]

クエリの結果の連続した各ローに対して、1 から開始する番号を生成します。NUMBER 関数は主に、SELECT リストで使用するために提供されています。

NUMBER 関数には制限があるため (以下の「備考」で説明します)、代わりに ROW_NUMBER 関数を使用します。ROW_NUMBER 関数には NUMBER と同じ機能ですが、NUMBER 関数にある制限はありません。次を参照してください。 「ROW_NUMBER 関数 [その他]」336 ページ.

構文

NUMBER([*])

戻り値

INT

備考

SELECT リストで NUMBER(*) を使用すると、結果セットのローに連番を付けることができます。NUMBER(*) は、各結果ローの ANSI ロー番号の値を返します。つまり、NUMBER 関数は、アプリケーションがどのように結果セットをスクロールするかに応じて、正または負の値を返します。insensitive カーソルの場合は、OPEN 時に結果セット全体が実体化されるため、NUMBER(*) は常に正の値を返します。

また、カーソルタイプによってはロー番号が変更される場合もあります。insensitive カーソルとスクロールカーソルの値は固定されています。同時更新を実行すると、動的カーソルと sensitive カーソルの値が変更される場合があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DELETE 文、WHERE 句、HAVING 句、ORDER BY 句、サブクエリ、集合を含むクエリ、いずれかの制約、GROUP BY 句、DISTINCT 句、集合演算子 (UNION、EXCEPT、INTERSECT)、派生テーブルで NUMBER 関数を使用すると、構文エラーになります。

NUMBER(*) は、ビューで使用できますが (前述の制限を受ける)、NUMBER(*) を伴う式に対応したビューのカラムは、クエリまたは外部ビューで多くても一度しか参照できません。また、ビューは左外部ジョインまたは全外部ジョインの NULL 入力テーブルとして使用できません。

Embedded SQL で、NUMBER(*) 関数があるクエリを参照するカーソルを使用する場合は、十分に注意してください。特に、データベースカーソルがカーソルの最後からの相対位置 (負のオフセットによる絶対位置) に配置されている場合、この関数は負の数を返します。

NUMBER は、UPDATE 文の SET 句の代入式の右側で使用できます。たとえば、SET x = NUMBER(*)。

また、SELECT 文から INSERT を使用すると、NUMBER 関数を使用してプライマリキーを生成できます。ただし、連続したプライマリキーの生成には、AUTOINCREMENT 句の使用をおすすめします。

AUTOINCREMENT 句の詳細については、「CREATE TABLE 文」651 ページを参照してください。

セマンティック上、NUMBER(*) と NUMBER() は同義です。

参照

- 「INSERT 文」802 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、連番が付けられた部署リストを返します。

```
SELECT NUMBER( * ), DepartmentName
FROM Departments
WHERE DepartmentID > 5
ORDER BY DepartmentName;
```

PATINDEX 関数 [文字列]

文字列のパターンが最初に出現した開始位置を表す整数を返します。

構文

PATINDEX('%*pattern***%'**, *string-expression* **)**

パラメーター

- ***pattern***　検索するパターン。先頭の % ワイルドカードを省略すると、PATINDEX 関数は、パターンが文字列の先頭に出現する場合は 1 を返し、それ以外の場合は 0 を返します。

パターンは、LIKE 比較と同じワイルドカードを使用します。次のワイルドカードがあります。

ワイルドカード	一致するもの
_ (アンダースコア)	任意の 1 文字
% (パーセント記号)	0 個以上の文字からなる任意の文字列
[]	指定範囲内、または一連の指定文字の任意の 1 文字
[^]	指定範囲外、または一連の指定文字以外の任意の 1 文字

- ***string-expression*** パターンを検索する文字列。

戻り値

INT

備考

PATINDEX 関数は、パターンが最初に出現した開始位置を返します。パターンが見つからない場合は、0 を返します。

参照

- 「LIKE 探索条件」42 ページ
- 「LOCATE 関数 [文字列]」271 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 2 を返します。

```
SELECT PATINDEX( '%hoco%', 'chocolate' );
```

次の文は、値 11 を返します。

```
SELECT PATINDEX( '%4_5_', '0a1A 2a3A 4a5A' );
```

次の文は、14 を返します。これは、文字列式の最初の英数字以外の文字です。データベースで大文字と小文字が区別されない場合は、'%[^a-zA-Z0-9]%' の代わりにパターン '%[^a-z0-9]%' を使用できます。

```
SELECT PATINDEX( '%[^a-zA-Z0-9]%', 'SQLAnywhere12 has many new features' );
```

文字列内の最初の英数字以外の文字を含めたそれまでのすべてを取得するには、次の手順に従います。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT LEFT( @string, PATINDEX( '%[^a-zA-Z0-9]%', @string ) );
```

次の文は、myTable というテーブルを作成し、英数字文字、スペース (ブランク)、英数字以外の文字が含まれる各種の文字列を入力します。続いて、SELECT 文とその後の結果により、PATINDEX を使用して文字列内のスペースと英数字以外の文字の開始位置を検索する方法がわかります。

```
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
thequickbrownfox jumpedoverthelazydog	0	0		thequickbrownfox jumpedoverthelazydog
thequickbrownfox jum999pedoverthelazydog	0	0		thequickbrownfox jum999pedoverthelazydog

col1	blank_posn	non_alpha_char	left_str	sub_str
thequick$$$ $brownfox	9	9	thequick$	$$$$brownfox
the quick brown fox$$ jumped over the lazy dog	4	20	the quick brown fox$	$$ jumped over the lazy dog

PERCENT_RANK 関数 [ランキング]

ロー X が関数の引数と ORDER BY 指定で定義される場合、PERCENT_RANK 関数は、グループのロー数で割ったロー X - 1 のランクを計算します。

構文

PERCENT_RANK( ) OVER (*window-spec* **)**

window-spec :次の「備考」の項を参照してください

戻り値

PERCENT_RANK 関数は、0 ～ 1 の DOUBLE 値を返します。

備考

window-spec の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。Window 関数として使用する場合、ORDER BY 句を指定する必要があります。また PARTITION BY 句を指定できますが、ROWS 句または RANGE 句は指定できません。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「CUME_DIST 関数 [ランキング]」191 ページ
- 「DENSE_RANK 関数 [ランキング]」213 ページ
- 「RANK 関数 [ランキング]」311 ページ

標準と互換性

- **SQL/2008**　PERCENT_RANK は、オプションの SQL/2008 言語機能 T612、「Advanced OLAP operations」の一部です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例は、ニューヨークの従業員の給与ランキングを性別ごとに示す結果セットを返します。結果は、降順にランキングされ、性別ごとに分けられます。

```
SELECT DepartmentID, Surname, Salary, Sex,
PERCENT_RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) "Rank"
FROM 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
100	Whitney	45700.000	F	0.666666667
400	Wetherby	35745.000	F	1

PI 関数 [数値]

PI の数値を返します。

構文

PI([*])

戻り値

DOUBLE

標準と互換性

- **SQL/2008** ベンダー拡張。

備考

この関数は DOUBLE 値を返します。

セマンティック上、PI(*) と PI() は同義です。

例

次の文は、値 3.141592653... を返します。

```
SELECT PI( * );
```

PLAN 関数 [その他]

SQL 文の長いプランの最適化方法を文字列で返します。

構文

PLAN(*string-expression*, [*cursor-type* [*update-status*]] **)**

パラメーター

- ***string-expression*** SQL 文。通常は SELECT 文ですが、UPDATE 文、MERGE 文、または DELETE 文も指定できます。
- ***cursor-type*** 文字列。*cursor-type* に使用できる値は asensitive (デフォルト)、insensitive、sensitive、または keyset-driven です。
- ***update-status*** 次のいずれかの値を受け入れる文字列パラメーター。これらの値は、指定されたカーソルをオプティマイザーがどのように処理するかを示します。

値	説明
READ-ONLY	このカーソルは読み込み専用。
READ-WRITE (デフォルト)	このカーソルは読み込みや書き込みが可能。
FOR UPDATE	このカーソルは読み込みや書き込みが可能。READ-WRITE とまったく同じです。

戻り値

LONG VARCHAR

参照

- 「高度：クエリ実行プラン」『SQL Anywhere サーバー SQL の使用法』
- 「EXPLANATION 関数 [その他]」228 ページ
- 「GRAPHICAL_PLAN 関数 [その他]」236 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、SELECT 文を文字列パラメーターとして渡し、クエリを実行するためのプランを返します。

```
SELECT PLAN(
  'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

この情報は、追加するインデックスの決定や、良いパフォーマンスを得るためのデータベース構造の決定に役立ちます。

次の文は、SELECT * FROM Departments WHERE DepartmentID > 100; クエリに対する INSENSITIVE カーソルのプランをテキスト形式で表した文字列を返します。

```
SELECT PLAN(
  'SELECT * FROM Departments WHERE DepartmentID > 100',
  'insensitive',
  'read-only' );
```

POWER 関数 [数値]

数のべき乗を表す数を計算します。

構文

POWER(*numeric-expression-1*, *numeric-expression-2* **)**

パラメーター

- ***numeric-expression-1*** べき乗の底。
- ***numeric-expression-2*** 指数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。引数のいずれかが NULL の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2008** POWER 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の文は、値 64 を返します。

```
SELECT POWER( 2, 6 );
```

PROPERTY_DESCRIPTION 関数 [システム]

プロパティの説明を返します。

構文

PROPERTY_DESCRIPTION({ *property-id* | *property-name* } **)**

パラメーター

- ***property-id*** データベースプロパティのプロパティ番号を表す整数。この番号は、PROPERTY_NUMBER 関数で調べることができます。*property-id* は、プロパティセットをループするときによく使われます。

- ***property-name*** データベースプロパティの名前を指定する文字列。

戻り値

VARCHAR

備考

各プロパティには、番号と名前の両方があります。ただし、番号はリリース間で変更されることがあるため、特定のプロパティの信頼できる識別子としては使用しないでください。

参照

- 「接続、データベース、データベースサーバーのプロパティ」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、インデックスの挿入数の説明を返します。

```
SELECT PROPERTY_DESCRIPTION( 'IndAdd' );
```

PROPERTY 関数 [システム]

指定したデータベースサーバーのプロパティの値を文字列で返します。

構文

PROPERTY({ *property-id* | *property-name* } [, *second-parameter*] **)**

パラメーター

- ***property-id*** データベースサーバープロパティのプロパティ番号を表す整数。この番号は、PROPERTY_NUMBER 関数で調べることができます。*property-id* は、プロパティセットをループするときによく使われます。

- ***property-name*** データベースプロパティの名前を指定する文字列。

- ***second-parameter*** 次に示すように、一部のプロパティには 2 つ目のパラメーターを指定できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

プロパティ	2 つ目のパラメーター	説明
EventTypeDesc	*positive-integer*	イベントタイプの説明を返すイベント ID を指定します。EventTypeDesc サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。
EventTypeName	*positive-integer*	イベントタイプ名を返すイベント ID を指定します。EventTypeName サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。
FunctionMaxParms	*positive-integer*	関数に指定できるパラメーターの最大数を返す関数番号を指定します。FunctionMaxParms サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。
FunctionMinParms	*positive-integer*	関数に指定する必要のあるパラメーターの最小数を返す関数番号を指定します。FunctionMinParms サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。
FunctionName	*positive-integer*	関数名を返す関数番号を指定します。FunctionName サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。

プロパティ	2 つ目のパラメーター	説明
Message	*positive-integer*	データベースサーバーメッセージウィンドウの対応する行の内容を返す行番号を指定します。行の先頭にはメッセージが表示された日付と時刻が追加されます。Message サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。
MessageText	*positive-integer*	データベースサーバーメッセージウィンドウ内の指定された行番号に対応するテキスト (日付と時刻を含まない) を返す行番号を指定します。MessageText サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。
MessageTime	*positive-integer*	データベースサーバーメッセージウィンドウ内の指定された行番号に対応する日付と時刻を返す行番号を指定します。MessageTime サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。
RemoteCapability	*positive-integer*	ID に対応するリモート機能名を返すリモート機能 ID を指定します。RemoteCapability サーバープロパティ『SQL Anywhere サーバー データベース管理』を参照してください。

戻り値

VARCHAR、LONG VARCHAR

備考

各プロパティには、番号と名前の両方があります。ただし、番号はリリース間で変更されることがあるため、特定のプロパティの信頼できる識別子としては使用しないでください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「DB_PROPERTY 関数 [システム]」208 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、現在のデータベースサーバー名を返します。

```
SELECT PROPERTY( 'Name' );
```

PROPERTY_NAME 関数 [システム]

指定した接続レベルで、指定したプロパティ ID を持つプロパティの名前を返します。

構文

PROPERTY_NAME(*property-id* **[,** *property-scope* **])**

property-scope:
NULL
| 'server'
| 'database'
| 'db'
| 'connection'
| 'conn'

パラメーター

- ***property-id*** データベースプロパティのプロパティ ID。
- ***property-scope*** プロパティのスコープまたは NULL。

戻り値

VARCHAR

参照

- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、プロパティ ID 102 に対応するサーバーレベルのプロパティを返します。

```
SELECT PROPERTY_NAME( 102, 'server' );
```

PROPERTY_NUMBER 関数 [システム]

指定したプロパティ名を持つプロパティのプロパティ番号を返します。

構文

PROPERTY_NUMBER(*property-name* **)**

パラメーター

- ***property-name*** プロパティ名。

戻り値

INT

備考

各プロパティには、番号と名前の両方があります。ただし、番号はリリース間で変更されることがあるため、特定のプロパティの信頼できる識別子としては使用しないでください。プロパティ番号またはプロパティ名をいずれかを使用できる場合は、プロパティ名を使用することをおすすめします。確実に現在サーバーで使用しているプロパティ番号にするために、常に PROPERTY_NUMBER を使用します。

参照

- 「接続、データベース、データベースサーバーのプロパティ」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、整数値を返します。実際の値は、リリースごとに変わります。

```
SELECT PROPERTY_NUMBER( 'PAGESIZE' );
```

QUARTER 関数 [日付と時刻]

指定した TIMESTAMP 式から、四半期を示す数を返します。

構文

QUARTER(*timestamp-expression* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***timestamp-expression*** 四半期にする日付。

戻り値

INTEGER

備考

四半期は次のとおりです。

四半期	期間 (開始日と終了日を含む)
1	1 月 1 日～ 3 月 31 日
2	4 月 1 日～ 6 月 30 日
3	7 月 1 日～ 9 月 30 日
4	10 月 1 日～ 12 月 31 日

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 2 を返します。

```
SELECT QUARTER( '1987/05/02' );
```

RADIANS 関数 [数値]

度数をラジアンに変換します。

構文

RADIANS(*numeric-expression* **)**

パラメーター

- ***numeric-expression*** 度数。この角度をラジアンに変換します。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、約 0.5236 の値を返します。

```
SELECT RADIANS( 30 );
```

RAND 関数 [数値]

オプションのシードから、間隔 0 ～ 1 の乱数を返します。

構文

RAND([*integer-expression*] **)**

パラメーター

- ***integer-expression***　乱数の作成に使用するオプションのシード。この引数を使用して、繰り返し可能な乱数列を作成できます。

戻り値

DOUBLE

備考

RAND 関数は、乗算線形合同法で乱数を生成します。詳細については、CACM 31 (10) の Park と Miller (1988) の寄稿 (1192 ～ 1201 ページ) と、Press 他 (1992) の『Numerical Recipes in C』(第 2 版の第 7 章 279 ページ、邦訳は『ニューメリカルレシピインシー日本語版—C 言語による数値計算のレシピ』) を参照してください。279). RAND 関数の呼び出し結果は、$0 < n < 1$ の疑似乱数 n です (0.0 と 1.0 はどちらも結果に含まれません)。

サーバーへの接続が確立するときに、この乱数生成関数は初期値のシードを指定します。各接続で異なる乱数シーケンスが生成されるように、各接続には一意のシードが指定されます。また、シード値 (*integer-expression*) を引数に指定することもできます。通常、シード値の指定は、以降の RAND 関数の呼び出しで乱数シーケンスを要求する前に 1 回だけ実行します。複数回シード値を初期化すると、シーケンスは再開されます。同じシード値を指定すると、同じシーケンスが生成されます。値が近いシード値の場合、似た初期シーケンスが生成されますが、シーケンスが進むにつれて相違が多くなります。

適切な乱数結果を取得するためでも、あるシード値から生成されたシーケンスと、別のシード値から生成されたシーケンスとは結合しないでください。つまり、乱数値のシーケンスを生成しているときに、シード値をリセットしないでください。

RAND 関数は非決定的関数として扱われます。クエリオプティマイザーは、RAND 関数の結果をキャッシュしません。

非決定的関数の詳細については、「関数のキャッシュ」『SQL Anywhere サーバー SQL の使用法』を参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は偶数の乱数結果を生成します。以降、シードを指定しないで RAND 関数を呼び出すと、毎回異なる結果が生成されます。

```
SELECT RAND( 1 );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
```

次の例では、シーケンスが同じ 2 つの結果セットが生成されます。これはシード値が 2 度指定されるためです。

```
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
```

次の例では、値が互いに近く、分布の点からは乱数とは言えない 5 つの結果が生成されます。このため、シード値が似た RAND 関数を複数回呼び出すことはおすすめしません。

```
SELECT RAND( 1 ), RAND( 2 ), RAND( 3 ), RAND( 4 ), RAND( 5 );
```

次の例は、5 つのまったく同じ結果が生成されるため、回避する必要があります。

```
SELECT RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 );
```

RANK 関数 [ランキング]

値のグループ内でのランクの値を計算します。同位の場合、RANK 関数はランキングシーケンス内にギャップを残します。

構文

RANK( ) OVER (*window-spec* **)**

window-spec : 以下の備考の説明を参照してください。

戻り値

INTEGER

備考

window-spec の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。Window 関数として使用する場合、ORDER BY 句を指定する必要があります。また PARTITION BY 句を指定できますが、ROWS 句または RANGE 句は指定できません。WINDOW 句の *window-spec* 定義を参照してください。「WINDOW 句」989 ページを参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「CUME_DIST 関数 [ランキング]」191 ページ
- 「DENSE_RANK 関数 [ランキング]」213 ページ
- 「ROW_NUMBER 関数 [その他]」336 ページ
- 「PERCENT_RANK 関数 [ランキング]」300 ページ

標準と互換性

- **SQL/2008** RANK 関数は、オプションの SQL/2008 言語機能 T612、「Advanced OLAP operations」の一部です。

例

次の例は、ユタとニューヨークの従業員の給与を降順にランキングします。7 番目と 8 番目の従業員は給与が同一であるため、どちらも 7 位にランキングされることに注意してください。これに続く従業員は 9 位にランキングされ、ランキングシーケンスにギャップが残されます (8 位のランキングはありません)。

```
SELECT Surname, Salary, State,
RANK() OVER (ORDER BY Salary DESC) "Rank"
FROM 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
...	...	...	...

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Surname	Salary	State	Rank
Lynch	24903.000	UT	19

READ_CLIENT_FILE 関数 [文字列]

クライアントコンピューター上で指定したファイルからデータを読み込みます。

構文

READ_CLIENT_FILE(*client-filename-expression* **)**

パラメーター

- ***client-filename-expression*** クライアントコンピューター上でのファイル名を示す CHAR 値。パスは、クライアントコンピューター上で、クライアントアプリケーションの現在の作業ディレクトリとの相対パスとして解決されます。

戻り値

LONG BINARY

備考

READ_CLIENT_FILE 関数が返す値は、指定されたクライアントファイルの内容を表します。BINARY 式が使用できる構文には、必ずこの関数を使用できます。

データはバイナリ文字列として返されるため、データが別の文字セットの場合、または圧縮または暗号化されている場合は、データの文字セットの変換、解凍、または復号化も必要となることがあります。

READ_CLIENT_FILE の評価中に、データベースサーバーは指定されたファイルのクライアントからの転送を開始します。クライアントは、転送要求を受信すると、クライアントファイルの共有ロックを取得し、データベースサーバーがクライアントに転送要求の終了を要求するまで、ロックを保持します。

クライアントのソフトウェアライブラリによってファイルの読み込みが行われ、Command Sequence 通信プロトコルを使用してデータの転送が実行されます。

パーミッション

クライアントコンピューターにあるファイルからデータを読み込む場合

●READCLIENTFILE 権限が必要です。

●読み込み元のディレクトリに対する読み込みパーミッションが必要です。

●allow_read_client_file データベースオプションが有効になっている必要があります。

●read_client_file セキュア機能が有効になっている必要があります。

参照

- 「allow_read_client_file オプション」『SQL Anywhere サーバー データベース管理』
- 「-sf dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「READCLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

参照

- 「クライアントコンピューター上のデータへのアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「READCLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』
- 「DECOMPRESS 関数 [文字列]」209 ページ
- 「DECRYPT 関数 [文字列]」211 ページ
- 「CSCONVERT 関数 [文字列]」189 ページ

REGEXP_SUBSTR 関数 [文字列]

正規表現を使用して文字列から部分文字列を抽出します。

構文

REGEXP_SUBSTR(*expression*,
regular-expression
[, *start-offset* [, *occurrence-number* [, *escape-expression*]]])

パラメーター

- ***expression***　検索される文字列。
- ***regular-expression***　一致させようとするパターン。正規表現の構文の詳細については、「正規表現の概要」17 ページを参照してください。
- ***start-offset***　検索を開始する *expression* へのオフセット。*start-offset* は、正の整数で表され、文字列の左端から数えた文字数を反映します。デフォルトは 1 (文字列の起点) です。
- ***occurrence-number***　*expression* 内で複数の一致がある場合に、検出する出現箇所を示す整数を指定します。たとえば、3 を指定すると、3 番目の出現箇所が検出されます。デフォルトは 1 です。
- ***escape-expression***　*regular-expression* に使用するエスケープ文字。デフォルトは円記号 (¥) です。

戻り値

LONG VARCHAR

備考

regular-expression が検出されない場合、REGEXP_SUBSTR は NULL を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

REGEXP 探索条件と同様に、REGEXP_SUBSTR 関数は一致と範囲評価にコードポイントを使用します。つまり、データベースの大文字と小文字の区別は結果に影響しません。
REGEXP_SUBSTR による一致とセット評価の実行方法については、「LIKE、REGEXP、SIMILAR TO：文字の比較における相違点」40 ページを参照してください。

部分文字クラスのみを含む文字クラスに一致させる場合は、外側の角カッコと、部分文字クラス用の角カッコを必ず使用してください (たとえば、REGEXP_SUBSTR (expression, '[[:digit:]]'))。部分文字クラスの一致の詳細については、「正規表現：特殊部分文字クラス」22 ページを参照してください。

start-offset が指定されている場合、そのオフセットは、一致させる式の開始を指定します。特に ^ は、*start-offset* で始まる式の先頭と一致します。

参照

- 「正規表現の構文」18 ページ
- 「REGEXP 探索条件」47 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。SQL/2008 標準で対応する関数は SUBSTRING_REGEX 関数であり、パラメーターがよく似ています。SUBSTRING_REGEX は、オプションの SQL/2008 言語機能 F844 の一部です。

例

次の例では、Employees.Street カラムの値を街路番号と街路名に分割します。

```
SELECT REGEXP_SUBSTR( Street, '^¥S+' ) as street_num,
 REGEXP_SUBSTR( Street, '(?<=^¥S+¥s+).*$' ) AS street_name
   FROM Employees;
```

street_num	street_name
9	East Washington Street
7	Pleasant Street
539	Pond Street
1244	Great Plain Avenue
...	...

現在の接続の IP アドレスが IP アドレスの範囲内 (この場合は、10.25.101.xxx または 10.25.102.xxx) かどうかを確認するには、次の文を実行します。

```
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;
```

REGR_AVGX 関数 [集合]

回帰直線の独立変数の平均を計算します。

構文 1

REGR_AVGX(*dependent-expression* , *independent-expression* **)**

構文 2

REGR_AVGX(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression*** 独立変数に影響される変数。
- ***independent-expression*** 結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。関数は、データの 1 回のパススルー中に同時に計算されます。NULL 値を除外した後で、次の計算が実行されます。式の *x* は *independent-expression* を表します。

```
AVG( x )
```

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「AVG 関数 [集合]」156 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SXY 関数 [集合]」326 ページ
- 「REGR_SYY 関数 [集合]」327 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ
- 「WINDOW 句」989 ページ

標準と互換性

- **SQL/2008**　REGR_AVGX は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、独立変数である従業員の年齢の平均を計算します。

```
SELECT REGR_AVGX( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_AVGY 関数 [集合]

回帰直線の従属変数の平均を計算します。

構文 1

REGR_AVGY(*dependent-expression* , *independent-expression* **)**

構文 2

REGR_AVGY(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression***　独立変数に影響される変数。
- ***independent-expression***　結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。関数は、データの 1 回のパススルー中に同時に計算されます。NULL 値を除外した後で、次の計算が実行されます。式の *y* は *dependent-expression* を表します。

```
AVG( y )
```

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SXY 関数 [集合]」326 ページ
- 「REGR_SYY 関数 [集合]」327 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「AVG 関数 [集合]」156 ページ
- 「WINDOW 句」989 ページ

標準と互換性

- **SQL/2008**　REGR_AVGY は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、独立変数である従業員の給与の平均を計算します。

```
SELECT REGR_AVGY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_COUNT 関数 [集合]

回帰直線の調整に使用される NULL 以外の数値のペアの数を表す整数を返します。

構文 1

REGR_COUNT(*dependent-expression* , *independent-expression* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 2

REGR_COUNT(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression*** 独立変数に影響される変数。
- ***independent-expression*** 結果に影響する変数。

戻り値

INTEGER

備考

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SXY 関数 [集合]」326 ページ
- 「REGR_SYY 関数 [集合]」327 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「COUNT 関数 [集合]」183 ページ
- 「AVG 関数 [集合]」156 ページ
- 「SUM 関数 [集合]」363 ページ

標準と互換性

- **SQL/2008** REGR_COUNT は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、回帰直線の調整に使用された NULL 以外のペアの数を返します。

```
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_INTERCEPT 関数 [集合]

従属変数と独立変数に最適な線形回帰直線の y 切片を計算します。

構文 1

REGR_INTERCEPT(*dependent-expression* , *independent-expression* **)**

構文 2

REGR_INTERCEPT(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression*** 独立変数に影響される変数。
- ***independent-expression*** 結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。関数は、データの 1 回のパススルー中に同時に計算されます。NULL 値を除外した後で、次の計算が実行されます。式の *y* は *dependent-expression* を表し、*x* は *independent-expression* を表します。

```
AVG( y ) - REGR_SLOPE( y, x ) * AVG( x )
```

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「WINDOW 句」989 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SXY 関数 [集合]」326 ページ
- 「REGR_SYY 関数 [集合]」327 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「AVG 関数 [集合]」156 ページ

標準と互換性

- **SQL/2008**　REGR_INTERCEPT は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、線形回帰直線の y 切片を返します。

```
SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_R2 関数 [集合]

回帰直線の決定係数 (**R-squared** または**適合度**の統計情報とも呼びます) を計算します。

構文 1

REGR_R2(*dependent-expression* , *independent-expression* **)**

構文 2

REGR_R2(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression***　独立変数に影響される変数。
- ***independent-expression***　結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SXY 関数 [集合]」326 ページ
- 「REGR_SYY 関数 [集合]」327 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ

標準と互換性

- **SQL/2008**　REGR_R2 は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、回帰直線の決定係数を返します。

```
SELECT REGR_R2( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SLOPE 関数 [集合]

NULL 以外のペアに調整された線形回帰直線の傾きを計算します。

構文 1

REGR_SLOPE(*dependent-expression* , *independent-expression* **)**

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 2

REGR_SLOPE(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression*** 独立変数に影響される変数。
- ***independent-expression*** 結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。関数は、データの 1 回のパススルー中に同時に計算されます。NULL 値を除外した後で、次の計算が実行されます。式の *y* は *dependent-expression* を表し、*x* は *independent-expression* を表します。

```
COVAR_POP( y, x ) / VAR_POP( x )
```

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SXY 関数 [集合]」326 ページ
- 「REGR_SYY 関数 [集合]」327 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ
- 「COVAR_POP 関数 [集合]」187 ページ
- 「VAR_POP 関数 [集合]」384 ページ

標準と互換性

- **SQL/2008**　REGR_SLOPE は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、値 935.3429749445614 を返します。

```
SELECT REGR_SLOPE( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SXX 関数 [集合]

線形回帰モデルに使用される独立した式の平方値の合計を返します。REGR_SXX 関数は、回帰モデルの統計的な有効性を評価するときに使用できます。

構文 1

REGR_SXX(*dependent-expression* , *independent-expression* **)**

構文 2

REGR_SXX(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression***　独立変数に影響される変数。
- ***independent-expression***　結果に影響する変数。

戻り値

DOUBLE

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。関数は、データの 1 回のパススルー中に同時に計算されます。NULL 値を除外した後で、次の計算が実行されます。式の *y* は *dependent-expression* を表し、*x* は *independent-expression* を表します。

```
REGR_COUNT( y, x ) * VAR_POP( x )
```

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ
- 「REGR_SXY 関数 [集合]」326 ページ
- 「REGR_SYY 関数 [集合]」327 ページ
- 「VAR_POP 関数 [集合]」384 ページ

標準と互換性

- **SQL/2008**　REGR_SXX は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、値 5916.4800000000105 を返します。

```
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SXY 関数 [集合]

従属変数と独立変数の積和を返します。REGR_SXY 関数は、回帰モデルの統計的な有効性を評価するときに使用できます。

構文 1

REGR_SXY(*dependent-expression* , *independent-expression* **)**

構文 2

REGR_SXY(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression*** 独立変数に影響される変数。
- ***independent-expression*** 結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行い、結果を DOUBLE で返します。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。関数は、データの 1 回のパススルー中に同時に計算されます。NULL 値を除外した後で、次の計算が実行されます。式の *y* は *dependent-expression* を表し、*x* は *independent-expression* を表します。

```
REGR_COUNT( y, x ) * COVAR_POP( y, x )
```

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「WINDOW 句」989 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SYY 関数 [集合]」327 ページ

標準と互換性

- **SQL/2008**　REGR_SXY は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、従属変数と独立変数の積和を返します。

```
SELECT REGR_SXY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SYY 関数 [集合]

回帰モデルの統計的有効性を評価できる値を返します。

構文 1

REGR_SYY(*dependent-expression* , *independent-expression* **)**

構文 2

REGR_SYY(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec :以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***dependent-expression***　独立変数に影響される変数。
- ***independent-expression***　結果に影響する変数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。関数が空のセットに適用される場合は、NULL を返します。

関数は、*dependent-expression* または *independent-expression* が NULL であるペアをすべて除外した後で、(*dependent-expression* と *independent-expression*) のペアのセットに適用されます。関数は、データの 1 回のパススルー中に同時に計算されます。NULL 値を除外した後で、次の計算が実行されます。式の *y* は *dependent-expression* を表し、*x* は *independent-expression* を表します。

```
REGR_COUNT( y, x ) * VAR_POP( y )
```

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_INTERCEPT 関数 [集合]」320 ページ
- 「REGR_COUNT 関数 [集合]」318 ページ
- 「REGR_AVGX 関数 [集合]」316 ページ
- 「REGR_AVGY 関数 [集合]」317 ページ
- 「REGR_SLOPE 関数 [集合]」322 ページ
- 「REGR_SXX 関数 [集合]」324 ページ
- 「REGR_SXY 関数 [集合]」326 ページ

標準と互換性

- **SQL/2008**　REGR_SYY は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の例は、値 26, 708, 672,843.3002 を返します。

```
SELECT REGR_SYY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REMAINDER 関数 [数値]

整数を整数で割ったときの余りを返します。

構文

REMAINDER(*dividend*, *divisor* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***dividend*** 被除数 (分数の分子)。
- ***divisor*** 除数 (分数の分母)。

戻り値

- INTEGER
- NUMERIC

備考

MOD 関数を使用して、余りを返すこともできます。

参照

- 「MOD 関数 [数値]」282 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 2 を返します。

```
SELECT REMAINDER( 5, 3 );
```

REPEAT 関数 [文字列]

文字列を指定された回数だけ連結します。

構文

REPEAT(*string-expression*, *integer-expression* **)**

パラメーター

- ***string-expression*** 繰り返される文字列。
- ***integer-expression*** 文字列を繰り返す回数。*integer-expression* が負の場合は、空の文字列を返します。

戻り値

- LONG VARCHAR
- LONG NVARCHAR

備考

実際の結果文字列の長さが戻り値の最大長を超えると、エラーが発生します。この場合、結果は文字列に使用できる最大サイズまでトランケートされます。

この関数の動作は、REPLICATE 関数と同じです。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「REPLICATE 関数 [文字列]」 331 ページ
- 「文字列関数」 148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 repeatrepeatrepeat を返します。

```
SELECT REPEAT( 'repeat', 3 );
```

REPLACE 関数 [文字列]

文字列を別の文字列で置換し、新しい結果を返します。

構文

REPLACE(*original-string*, *search-string*, *replace-string* **)**

パラメーター

いずれかの引数が NULL の場合、NULL を返します。

- ***original-string***　検索される文字列。任意の長さの文字列を指定できます。
- ***search-string***　検索して *replace-string* に置換する文字列。この文字列は 255 バイトに制限されています。*search-string* が空の文字列の場合は、元の文字列を未変更のまま返します。
- ***replace-string***　*search-string* と置き換える置換文字列。任意の長さの文字列を指定できます。*replacement-string* が空の文字列の場合は、すべての *search-string* が削除されます。

戻り値

- LONG VARCHAR
- LONG NVARCHAR

備考

この関数はすべてのオカレンスを置換します。

大文字と小文字を区別するデータベースでの比較では、大文字と小文字が区別されます。

この関数は NCHAR の入力または出力をサポートしています。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SUBSTRING 関数 [文字列]」361 ページ
- 「CHARINDEX 関数 [文字列]」169 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 xx.def.xx.ghi を返します。

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' );
```

次の文は、ALTER PROCEDURE 文を含む結果セットを生成します。ALTER PROCEDURE を実行すると、名前が変更されたテーブルを参照する格納済みプロシージャーが修復されます (テーブル名を一意にすることをおすすめします)。

```
SELECT REPLACE(
  REPLACE( proc_defn, 'OldTableName', 'NewTableName' ),
  'CREATE PROCEDURE',
  'ALTER PROCEDURE')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%';
```

REPLICATE 関数 [文字列]

文字列を指定された回数だけ連結します。

構文

REPLICATE(*string-expression*, *integer-expression* **)**

パラメーター

- ***string-expression***　繰り返される文字列。
- ***integer-expression***　文字列を繰り返す回数。

戻り値

- LONG VARCHAR
- LONG NVARCHAR

備考

実際の結果文字列の長さが戻り値の最大長を超えると、エラーが発生します。この場合、結果は文字列に使用できる最大サイズまでトランケートされます。

この関数の動作は、REPEAT 関数と同じです。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「REPEAT 関数 [文字列]」329 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 repeatrepeatrepeat を返します。

```
SELECT REPLICATE( 'repeat', 3 );
```

REVERSE 関数 [文字列]

文字式の逆の値を返します。

構文

REVERSE(*string-expression* **)**

パラメーター

- ***string-expression***　逆転される文字列。

戻り値

- LONG VARCHAR
- LONG NVARCHAR

備考

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 cba を返します。

```
SELECT REVERSE( 'abc' );
```

REWRITE 関数 [その他]

書き換えられた SELECT 文、UPDATE 文または DELETE 文を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

REWRITE(*select-statement* **[, 'ANSI'])**

パラメーター

- ***select-statement*** 関数の結果を生成するために書き換えの最適化を適用する SQL 文。

戻り値

LONG VARCHAR

備考

ANSI 引数を指定せずに REWRITE 関数を使用すると、オプティマイザーがどのように特定のクエリのアクセスプランを生成したかを理解できます。特に、文の WHERE 句、ON 句、HAVING 句の条件が SQL Anywhere によってどのように書き換えられたかを確認して、要求の実行時間を改善するために適用できるインデックスの有無を特定することができます。

REWRITE から返された文は、元の文のセマンティックと一致しない場合があります。これは、一部のリライト最適化では、SQL に直接変換できない内部メカニズムが導入されるためです。たとえば、サーバーではローの識別子を使用して重複を排除しますが、これは SQL に変換できません。

REWRITE 関数によって書き換えられたクエリは、実行するためのクエリではありません。リライトフェーズの後に実際にオプティマイザーに何が渡されるかを示すことで、パフォーマンス問題を分析するためのツールです。

REWRITE の出力に反映されないリライト最適化もあります。それらは、LIKE 最適化、minimum 関数または maximum 関数の最適化、上限／下限の排除、述部の仮定条件などです。

ANSI を指定すると、REWRITE は文に相当する ANSI を返します。この場合は、次のリライト最適化のみが適用されます。

●Transact-SQL 外部ジョインが、ANSI SQL 外部ジョインに書き換えられる

●重複した相関名が削除される

●KEY ジョインと NATURAL ジョインが ANSI SQL ジョインに書き換えられる

参照

- 「クエリ処理中に実行される最適化」『SQL Anywhere サーバー SQL の使用法』
- 「extended_join_syntax オプション」『SQL Anywhere サーバー データベース管理』
- 「Transact-SQL の外部ジョイン (*= or =*)」『SQL Anywhere サーバー SQL の使用法』
- 「キージョイン」『SQL Anywhere サーバー SQL の使用法』
- 「ナチュラルジョイン」『SQL Anywhere サーバー SQL の使用法』
- 「ジョインで重複する相関名 (スタージョイン)」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、クエリでリライト最適化を 2 回実行します。最初の最適化では、サブクエリのネストを解除して、Employees テーブルと SalesOrders テーブルをジョインします。2 回目の最適化では、Employees と SalesOrders の間のプライマリキーと外部キーのジョインを削除してクエリを簡素化します。このリライト最適化の一部では、ジョインの述部 e.EmployeeID=s.SalesRepresentative が述部 s.SalesRepresentative IS NOT NULL に置換されます。

```
SELECT REWRITE( 'SELECT s.ID, s.OrderDate
   FROM SalesOrders s
   WHERE EXISTS ( SELECT *
      FROM Employees e
         WHERE e.EmployeeID = s.SalesRepresentative)' ) FROM dummy;
```

このクエリは、書き換えられたクエリがある単一カラムの結果セットを返します。

```
'SELECT s.ID, s.OrderDate FROM SalesOrders s WHERE s.SalesRepresentative IS NOT NULL'
```

次の REWRITE の例は、ANSI 引数を使用します。

```
SELECT REWRITE( 'SELECT DISTINCT s.ID, s.OrderDate, e.GivenName, e.EmployeeID
   FROM SalesOrders s, Employees e
         WHERE e.EmployeeID *= s.SalesRepresentative', 'ANSI' ) FROM dummy;
```

結果は、この文に相当する ANSI です。この場合は、Transact-SQL 外部ジョインが ANSI 外部ジョインに変換されます。このクエリは、単一カラムの結果セットを返します (読みやすいように改行しています)。

```
'SELECT DISTINCT s.ID, s.OrderDate, e.GivenName, e.EmployeeID
 FROM Employees as e
 LEFT OUTER JOIN  SalesOrders as s
 ON e.EmployeeID = s.SalesRepresentative';
```

RIGHT 関数 [文字列]

文字列の一番右側にある文字を返します。

構文

RIGHT(*string-expression*, *integer-expression* **)**

パラメーター

- ***string-expression*** 一番右側にある文字を返す文字列。
- ***integer-expression*** 返される文字列の末尾の文字数。

戻り値

- LONG VARCHAR
- LONG NVARCHAR

備考

文字列にマルチバイト文字があり、適切な照合が使用されている場合、返されるバイト数が指定した文字数よりも大きい場合があります。

カラムの値より大きな *integer-expression* を指定できます。この場合、全体の値が返されます。

この関数は NCHAR の入力または出力をサポートしています。入力文字が文字長のセマンティックを使用している場合、可能であれば、戻り値が文字長のセマンティックで記述されます。

参照

- 「LEFT 関数 [文字列]」265 ページ
- 「国際言語と文字セット」『SQL Anywhere サーバー データベース管理』
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、Customers テーブルに含まれる各 Surname 値の最後の 5 文字を返します。

```
SELECT RIGHT( Surname, 5) FROM Customers;
```

ROUND 関数 [数値]

integer-expression で指定した小数点以下の桁数に *numeric-expression* を丸めます。

構文

ROUND(*numeric-expression*, *integer-expression* **)**

パラメーター

- ***numeric-expression***　関数に渡される、丸めの対象となる数。
- ***integer-expression***　正の整数は、丸めを行う小数点の右側の有効桁数を指定します。負の式は、丸めを行う小数点の左側の有効桁数を指定します。

戻り値

NUMERIC

備考

この関数の結果は numeric または double です。数値の結果があり、整数 *integer-expression* が負の値の場合、精度は 1 ずつ増えます。

参照

- 「TRUNCNUM 関数 [数値]」376 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 123.200 を返します。

```
SELECT ROUND( 123.234, 1 );
```

ROW_NUMBER 関数 [その他]

各ローにユニークな番号を割り当てます。この関数は NUMBER 関数の代わりに使用できます。

構文

ROW_NUMBER( ) OVER (*window-spec* **)**

window-spec :次の「備考」の項を参照してください

戻り値

INTEGER

備考

window-spec の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。Window 関数として使用する場合、ORDER BY 句を指定する必要があります。また PARTITION BY 句を指定できますが、ROWS 句または RANGE 句は指定できません。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「NUMBER 関数 [その他]」296 ページ
- 「RANK 関数 [ランキング]」311 ページ
- 「ROWID 関数 [その他]」337 ページ

標準と互換性

- **SQL/2008** ROW_NUMBER は、オプションの SQL/2008 言語機能 T611、「Elementary OLAP operations」の一部です。

例

次の例は、ニューヨークとユタの各従業員のユニークなロー番号を示す結果セットを返します。クエリは Salary の降順に配列されるため、最初のロー番号は、データセット内で最も給与の高い従業員に割り当てられます。2 人の従業員の給与が同一ですが、2 人の従業員にはユニークなロー番号が割り当てられるため、同順は解決されません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT Surname, Salary, State,
ROW_NUMBER() OVER (ORDER BY Salary DESC) "Rank"
FROM 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

ROWID 関数 [その他]

テーブル内のローを一意に識別する UNSIGNED BIGINT ビット値を返します。

構文

ROWID(*correlation-name* **)**

パラメーター

- ***correlation-name*** クエリで使用されるテーブルの相関名。相関名は、ベーステーブル、テンポラリテーブル、グローバルテンポラリテーブル、またはプロキシテーブル (基本となるプロキシサーバーが同様の機能をサポートしている場合にのみ使用できます) を指す必要があります。ROWID 関数の引数では、ビュー、派生テーブル、共通のテーブル式、またはプロシージャーを参照しないでください。

戻り値

UNSIGNED BIGINT

備考

指定された相関名に対応するテーブル内のローの、ロー識別子を返します。

この関数によって返される値は、複数のクエリ間で一貫しているとはかぎりません。データベースで実行されるさまざまな演算の結果、テーブルのロー識別子が変更される可能性があるためです。特に、REORGANIZE TABLE 文を実行すると、ロー識別子が変更される可能性があります。また、ローが削除された後にロー識別子が再利用される場合もあります。そのため、通常は ROWID 関数の使用を避け、代わりにプライマリキー値に基づいてロー識別子を取得してください。ROWID は診断を実行する場合にのみ使用することをおすすめします。

関数の結果は UNSIGNED BIGINT ですが、この値について数学演算を行っても意味がありません。たとえば、ロー識別子に 1 を追加しても、次のローのロー識別子にはなりません。また、ROWID を使用する場合、等号と IN 述部のみを検索引数にすることができます。必要に応じて、ROWID を使用する述部 (ROWID(T) = literal など) は 64 ビット UNSIGNED INTEGER 値へのキャストに使用できます。変換を実行できない場合、データの例外処理が発生します。*literal* の値が無効なロー識別子の場合、比較述部は FALSE と評価されます。

ROWID 関数をテーブルまたはカラムの検査制約内で使用することはできません。また、計算カラムの COMPUTE 式でも使用できません。

参照

- 「ROW_NUMBER 関数 [その他]」336 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、Employee のローから、105 のロー識別子を返します。

```
SELECT ROWID( Employees ) FROM Employees WHERE Employees.EmployeeID = 105;
```

次の文は、Employees テーブルのローのロックリストとそのローの内容を返します。

```
SELECT *
  FROM sa_locks() S JOIN Employees WITH( NOLOCK )
    ON ROWID( Employees ) = S.row_identifier
  WHERE S.table_name = 'Employees';
```

RTRIM 関数 [文字列]

文字列の後続ブランクを削除します。

構文

RTRIM(*string-expression* **)**

パラメーター

- ***string-expression***　削除する文字列。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

備考

結果の実際の長さは、式の長さから、削除する文字数を引いた値です。すべての文字を削除すると、結果は空の文字列になります。

引数が NULL の場合、結果は NULL 値になります。

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「TRIM 関数 [文字列]」375 ページ
- 「LTRIM 関数 [文字列]」274 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

  SQL/2008 標準で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数で提供されます。

例

次の文は、すべての後続ブランクが削除された文字列 Test Message を返します。

```
SELECT RTRIM( 'Test Message     ' );
```

SECOND 関数 [日付と時刻]

TIMESTAMP 引数の秒の値を返します。

構文

SECOND(*timestamp-expression* **)**

パラメーター

- ***timestamp-expression***　TIMESTAMP 値。

戻り値

SMALLINT

備考

指定した TIMESTAMP 引数値の秒に相当する 0 ～ 59 の数を返します。

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 25 を返します。

```
SELECT SECOND( '1998-07-13 21:21:25' );
```

SECONDS 関数 [日付と時刻]

SECONDS 関数は、TIMESTAMP を操作するか、2 つの TIMESTAMP 値の間の秒数を返します。次の「備考」の項を参照してください。

構文 1

SECONDS(*timestamp-expression* **)**

構文 2

SECONDS(*timestamp-expression*, *timestamp-expression* **)**

構文 3

SECONDS(*time-or-timestamp-expression*, *integer-expression* **)**

パラメーター

- ***timestamp-expression***　TIMESTAMP 値。
- ***time-or-timestamp-expression***　TIME または TIMESTAMP データ型の値。
- ***integer-expression***　*time-or-timestamp-expression* に加算される秒数。*integer-expression* が負の場合、*time-or-timestamp-expression* から適切な秒数が引かれます。整数式を指定する場合は、*time-or-timestamp-expression* を TIME、DATE または TIMESTAMP データ型として明示的にキャストしてください。*time-or-timestamp-expression* が DATE データ型の場合、時刻部分には真夜中が使用されます。

戻り値

構文 1 では UNSIGNED BIGINT。

構文 2 では SIGNED BIGINT。

構文 3 では TIME または TIMESTAMP。

備考

SECONDS 関数の結果は、その引数によって異なります。

- **構文 1**　1 つの *timestamp-expression* を SECONDS 関数に渡すと、0000-02-29 の真夜中から *timestamp-expression* までの秒数を UNSIGNED BIGINT として返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **注意**
> 0000-02 は実際の日付を指すための値ではありません。SECONDS 関数で使用されるデフォルトの日付です。

- **構文 2**　2 つの TIMESTAMP 値を SECONDS 関数に渡すと、その間の秒数を整数の SIGNED BIGINT 値で返します。

- **構文 3**　TIMESTAMP 値と INTEGER 値を SECONDS 関数に渡すと、整数の秒数を *time-or-timestamp-expression* に加算した TIMESTAMP 結果を返します。同様に、TIME 値を SECONDS 関数に渡すと、TIME データ型の値を返します。

構文 2 の代わりに、DATEDIFF 関数を使用します。構文 3 の代わりに、DATEADD 関数を使用します。

参照

- 「CAST 関数 [データ型変換]」 165 ページ
- 「DATEADD 関数 [日付と時刻]」 194 ページ
- 「DATEDIFF 関数 [日付と時刻]」 196 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 14400 を返します。これは、2 番目の TIMESTAMP 値が、最初のタイムスタンプの 14400 秒後であることを示します。

```
SELECT SECONDS( '1999-07-13 06:07:12',
  '1999-07-13 10:07:12' );
SELECT DATEDIFF( second,
  '1999-07-13 06:07:12',
  '1999-07-13 10:07:12' );
```

次の文は、値 63062431632 を返します。

```
SELECT SECONDS( '1998-07-13 06:07:12' );
```

次の文は、TIMESTAMP 値 1999-05-12 21:05:12.000 を返します。

```
SELECT SECONDS( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5);
SELECT DATEADD( second, 5, '1999-05-12 21:05:07' );
```

SET_BIT 関数 [ビット配列]

ビット配列の特定ビットの値を設定します。

構文

SET_BIT([*bit-expression*,]*bit-position* [, *value*] **)**

パラメーター

- ***bit-expression*** ビットを変更するビット配列。
- ***bit-position*** 設定するビットの位置。これは符号なしの整数にしてください。
- ***value*** ビットに設定する値。

戻り値

LONG VARBIT

備考

bit-expression のデフォルト値は、すべてのビットが 0 (FALSE) に設定された長さ *bit-position* のビット配列です。

value のデフォルト値は 1 (TRUE) です。

いずれかのパラメーターが NULL の場合、結果は NULL です。

配列の位置は左側からカウントします。初期値は 1 です。

参照

- 「GET_BIT 関数 [ビット配列]」233 ページ
- 「SET_BITS 関数 [集合]」343 ページ
- 「INTEGER データ型」99 ページ
- 「ビット処理演算子」11 ページ
- 「sa_get_bits システムプロシージャー」1086 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 00100011 を返します。

```
SELECT SET_BIT( '00110011', 4 , 0);
```

次の文は、値 00111011 を返します。

```
SELECT SET_BIT( '00110011', 5 , 1);
```

次の文は、値 00111011 を返します。

```
SELECT SET_BIT( '00110011', 5 );
```

次の文は、値 00001 を返します。

```
SELECT SET_BIT( 5 );
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SET_BITS 関数 [集合]

ローのセットに含まれる値に対応する特定ビットを 1 (TRUE) に設定するときに、ビット配列を作成します。

構文

SET_BITS(*expression* **)**

パラメーター

- ***expression*** 1 に設定するビットを決定するときに使用する式。これは一般的にカラム名です。

戻り値

LONG VARBIT

備考

指定した値が NULL のローは無視されます。

ローがない場合、NULL が返されます。

結果の長さは、1 に設定された最大の位置です。

SET_BITS 関数も次の文と同様ですが、より高速です。

```
SELECT BIT_OR( SET_BIT( expression ) )
FROM table;
```

参照

- 「ビット処理演算子」11 ページ
- 「GET_BIT 関数 [ビット配列]」233 ページ
- 「SET_BIT 関数 [ビット配列]」341 ページ
- 「sa_get_bits システムプロシージャー」1086 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、2 番目、5 番目、10 番目の各ビットが 1 に設定されたビット配列 (つまり 0100100001) を返します。

```
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;
```

SIGN 関数 [数値]

指定された数値の符号 (正または負) を返します。

構文

SIGN(*numeric-expression* **)**

パラメーター

- ***numeric-expression*** 符号が返される数値。*numeric-expression* には、INTEGER、DOUBLE、NUMERIC のデータ型を使用できます。

戻り値

SMALLINT

備考

負の数を指定すると、SIGN 関数は -1 を返します。

0 を指定すると、SIGN 関数は 0 を返します。

正の数を指定すると、SIGN 関数は 1 を返します。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 -1 を返します。

```
SELECT SIGN( -550 );
```

SIMILAR 関数 [文字列]

2 つの文字列の類似性を示す数を返します。

構文

SIMILAR(*string-expression-1*, *string-expression-2* **)**

パラメーター

- ***string-expression-1*** 比較する最初の文字列。
- ***string-expression-2*** 比較する 2 番目の文字列。

戻り値

SMALL INT

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

SIMILAR 関数は、2 つの文字列の類似性を表す 0 ～ 100 の整数を返します。結果は、2 つの文字列の文字が一致している割合として解釈できます。値が 100 の場合は、2 つの文字列は同じです。

この関数を使用して、名前 (顧客名など) のリストを修正できます。顧客がわずかに異なる名前で重複して登録されている場合があります。SIMILAR 関数を使用して、類似する顧客名を見つけることができます。このことを行うには、customer テーブルをそのテーブル自体にジョインし、類似性が 90 パーセントより大きく、かつ 100 パーセント未満のすべての顧客のレポートを作成します。

SIMILAR 関数で実行される計算は、単に文字数が一致する場合よりも複雑になります。

参照

● 「文字列関数」 148 ページ

標準と互換性

● **SQL/2008**　ベンダー拡張。

例

次の文は値 75 を返します。2 つの値が 75 % 前後であることを示します。

```
SELECT SIMILAR( 'toast', 'coast' );
```

SIN 関数 [数値]

数のサインを返します。

構文

SIN(*numeric-expression* **)**

パラメーター

● ***numeric-expression***　角度 (ラジアン)。

戻り値

DOUBLE

備考

SIN 関数は、引数のサインを返します。この引数はラジアン単位で表現される角度です。SIN 関数と ASIN 関数は逆変換の演算です。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「ASIN 関数 [数値]」 154 ページ
- 「COS 関数 [数値]」 182 ページ
- 「COT 関数 [数値]」 183 ページ
- 「TAN 関数 [数値]」 367 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、0.52 の SIN 値を返します。

```
SELECT SIN( 0.52 );
```

SOAP_HEADER 関数 [SOAP]

SOAP ヘッダーエントリまたは SOAP 要求のヘッダーエントリの属性値を返します。

構文

SOAP_HEADER(*header-key* **[**, *index*, *header-attribute* **])**

パラメーター

- ***header-key***　VARCHAR パラメーターには、特定の SOAP ヘッダーエントリで最上位層にある XML 要素の XML ローカル名を指定します。

- ***index***　このオプションの INTEGER パラメーターは、同じ名前の SOAP ヘッダーフィールドでも違いを見つけます。同じ名前になる状況は、複数のヘッダーエントリが同じ localname を持つ最上位 XML 要素がある場合に発生します。通常、このような要素には一意の番号空間があります。

- ***header-attribute***　オプションの VARCHAR パラメーターは、次に示すヘッダーエントリ要素内にある任意の属性ノードを指定できます。

  - **@namespace**　特定のヘッダーエントリの名前空間にアクセスするときに使用する特殊な SQL Anywhere 属性。

  - **mustUnderstand**　ヘッダーエントリが必須かオプションかを処理する受信者に示す SOAP 1.1 のヘッダーエントリ属性。

  - **encodingStyle**　エンコーディングスタイルを示す SOAP 1.1 ヘッダーエントリ属性。この属性にはアクセスできますが、SQL Anywhere の内部では使用されません。

  - **actor**　受信者の URL を指定することで、ヘッダーエントリの受信者を示す SOAP 1.1 ヘッダーエントリ属性。

戻り値

LONG VARCHAR

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

この関数は、単一のパラメーター *header-key* を使用してヘッダーエントリを返すときに使用します。ヘッダーエントリは、SOAP ヘッダーに含まれている要素と、その要素に含まれるすべての部分要素を XML 文字列で表現したものです。

この関数は、オプションの *index* と *header-attribute* パラメーターを指定して、ヘッダーエントリ属性を抽出するときにも使用できます。

この関数は、指定された SOAP ヘッダーフィールドの値を返します。SOAP サービスから呼び出されていない場合は NULL を返します。Web サービスを介して SOAP 要求を処理する場合に使用します。

指定した *header-key* のヘッダーが存在しない場合、戻り値は NULL です。

参照

- 「NEXT_SOAP_HEADER 関数 [SOAP]」293 ページ
- 「sa_set_soap_header システムプロシージャー」1190 ページ
- 「チュートリアル：SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

HTTP Web サービスによって呼び出されるストアドプロシージャー内で使用された場合、次の例は SOAP 要求ヘッダーにあるすべてのキーを処理します。**Authentication** キーを処理するときに、キーの値も取得します。

```
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;
```

SORTKEY 関数 [文字列]

ソートキー値を生成します。つまり、代替照合規則に基づいて文字列をソートする場合に使用できる値を生成します。

構文

SORTKEY(*string-expression*
[, { *collation-id*
| *collation-name*[**(** *collation-tailoring-string*)] }]
)

パラメーター

- ***string-expression*** 文字列式には、データベース側文字セットでエンコードされた文字のみを指定できます。

  string-expression が空の文字列の場合、SORTKEY 関数は長さ 0 のバイナリ値を返します。*string-expression* が NULL の場合、SORTKEY 関数は NULL 値を返します。空の文字列のソート順の値は、データベースカラムの NULL 文字列の値とは異なります。

  SORTKEY 関数が処理できる文字列の最大長は 254 バイトです。これより長い部分は、無視されます。

- ***collation-name*** 使用するソート順の名前を指定する文字列または文字変数。また、alias char_collation または equivalently, db_collation を指定して、データベースが CHAR の照合に使用するソートキーを生成するときに指定することもできます。同様に、エイリアス nchar_collation を指定して、データベースで使用している NCHAR の照合順でソートキーを生成できます。

- ***collation-id*** 使用するソート順の ID 番号を指定する変数、定数 (整数)、または文字列。*collation-id* を指定しない場合のデフォルトは、デフォルト Unicode マルチ言語です。

- ***collation-tailoring-string*** オプションで、文字列のソートや比較を詳細に制御することを目的に、照合の適合化オプション (*collation-tailoring-string*) を指定できます。これらのオプションは、キーワード=値 の形式で、カッコで囲んで指定して、その後ろに照合名を記述します。たとえば、'UCA(locale=es;case=LowerFirst;accent=respect)' のように記述します。これらのオプションの組み合わせを指定する構文は、CREATE DATABASE 文の COLLATION 句を定義する構文と同じです。

> **注意**
> UCA 照合を指定すると、照合の適合化のすべてのオプションがサポートされます。その他の照合の場合、大文字小文字の区別の適合化のみがサポートされます。

戻り値

BINARY

備考

SORTKEY 関数が生成する値を使用して、事前定義済みのソート順の動作に基づいて結果を順序付けることができます。これにより、データベース照合では使用できない文字ソート順の動作を操作できます。SORTKEY 関数から保持される値はバイナリ値で、入力文字列のエンコードされたソート順情報が含まれています。たとえば、SORTKEY 関数から返された値を、ソース文字列と一緒にカラムに格納できます。必要な順序で文字データを取り出すには、SORTKEY 関数の実行結果が格納されているカラムの SELECT 文に ORDER BY 句を指定するだけです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SORTKEY 関数は、特定のソート順の基準セットに対して返された値が、varbinary データ型で実行されるバイナリ比較で使用できることを保証します。

クエリのソートキーを作成する作業には負荷がかかります。使用頻度の高いソートキーの代替手段として、ソートキーの値を保存する計算カラムを作成し、クエリの ORDER BY 句を使用してカラムを参照する方法を検討します。

SORTKEY 関数の入力は、各入力文字に対して最大 6 バイトのソート順情報を生成できます。SORTKEY 関数の出力は VARBINARY 型で、最大長は 1024 バイトです。

ソートキーの生成中に照合に UCA を指定すると、デフォルトでは、照合の適合化はアクセント記号および大文字と小文字が区別されます。たとえば、UCA が自身によって指定されると、適用されるデフォルトの適合化は 'UCA(case=UpperFirst;accent=Respect;punct=Primary)' に等しくなります。

SORTKEY に対する 2 番目のパラメーターで別の照合を指定すると、その設定内容によって、デフォルトの設定内容が上書きされます。たとえば、次の 2 つの文は同じです。

```
SELECT SORTKEY( 'abc', 'UCA(accent=Ignore)' );
SELECT SORTKEY( 'abc', 'UCA(case=UpperFirst;accent=Ignore;punct=Primary)' );
```

UCA 以外の照合を指定しても、デフォルトでは、照合の適合化はアクセント記号および大文字と小文字が区別されます。ただし、UCA 以外の照合の場合、照合の適合化を使用して上書きできるのは、大文字と小文字の区別のみです。次に例を示します。

```
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

適合化オプションを指定しないで作成されたデータベースでは (たとえば、dbinit -c -zn uca mydb.db)、次の 2 つの句では異なるソート順が生成されることがあります。これは、SORTKEY 関数に対してデータベースの照合名が指定されている場合でも同様です。

```
ORDER BY string-expression
ORDER BY SORTKEY( string-expression, database-collation-name )
```

このような現象が起こるのは、データベースの作成に使用されたデフォルトの適合化設定と、SORTKEY 関数のデフォルトの適合化設定が異なるためです。SORTKEY でもデータベース照合と同じ動作が実行されるようにするには、データベース照合の設定に一致する *collation-tailoring-string* の調整構文を指定するか、*collation-name* に対して db_collation を指定します。次に例を示します。

```
SORTKEY( expression, 'db_collation' )
```

注意

SQL Anywhere のバージョンに応じて、異なるソートキー値が生成されます。このため、あるバージョンの SQL Anywhere で作成されたソートキー値を別のバージョンの SQL Anywhere で作成されたデータベースで使用すると、ソートに関する問題が発生する場合があります。この問題が発生した場合は、ソートキー値を再生成してください。

アンロード／再ロードを使用してデータベースをアップグレードする場合も、ソートキー値を再生成してください。

参照

- 「sort_collation オプション」『SQL Anywhere サーバー データベース管理』
- 「サポートされている照合と代替照合」『SQL Anywhere サーバー データベース管理』
- 「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』
- 「推奨文字セットと照合」『SQL Anywhere サーバー データベース管理』
- 「COMPARE 関数 [文字列]」171 ページ
- 「CREATE DATABASE 文」521 ページ
- 「国際言語と文字セット」『SQL Anywhere サーバー データベース管理』
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、Employees テーブルに問い合わせ、すべての従業員の FirstName と Surname を返します。結果は、dict 照合 (Latin-1、英語、フランス語、ドイツ語辞書) を使用して、Surname カラムのソートキー値でソートされます。

```
SELECT Surname, GivenName FROM Employees ORDER BY SORTKEY( Surname, 'dict' );
```

次の例は、UCA 照合と適合化オプションを使用して、abc のソートキー値を返します。

```
SELECT SORTKEY( 'abc', 'UCA(locale=es;case=LowerFirst;accent=respect)' );
```

SOUNDEX 関数 [文字列]

文字列の発音を表す数を返します。

構文

SOUNDEX(*string-expression* **)**

パラメーター

- ***string-expression***　評価される文字列。

戻り値

SMALLINT

備考

文字列の SOUNDEX 関数の値は、先頭の文字とそれに続く H、Y、W 以外の 3 つの子音がベースになっています。文字列の最初の文字である場合を除き、*string-expression* の母音は無視されます。同じ文字が 2 つ続く場合は 1 文字としてカウントされます。たとえば、"apples" という単語は、文字 A、P、L、S がベースになります。

マルチバイト文字は、SOUNDEX 関数では無視されます。

完全とは言えませんが、SOUNDEX 関数は通常、類似発音で同じ文字で始まる語句に対して同じ数を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SOUNDEX 関数は、英単語の処理に最も優れています。他の言語の処理は英語の場合よりも劣ります。

参照

- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、それぞれの名前の発音を表す 2 つの同じ数 3827 を返します。

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

SPACE 関数 [文字列]

指定した数のスペースを返します。

構文

SPACE(*integer-expression* **)**

パラメーター

- ***integer-expression*** 返すスペースの数。

戻り値

LONG VARCHAR

備考

integer-expression が負の場合は、NULL 文字列を返します。

参照

- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、10 のスペースから成る文字列を返します。

```
SELECT SPACE( 10 );
```

SQLDIALECT 関数 [その他]

文の SQL ダイアレクトを示す Watcom SQL または Transact-SQL のどちらかを返します。

構文

SQLDIALECT(*sql-statement-string* **)**

パラメーター

- ***sql-statement-string*** 関数がダイアレクトの判断に使用する SQL 文。

戻り値

LONG VARCHAR

参照

- 「TRANSACTSQL 関数 [その他]」373 ページ
- 「WATCOMSQL 関数 [その他]」388 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、文字列 Transact-SQL を返します。

```
SELECT
  SQLDIALECT( 'SELECT EmployeeName = Surname FROM Employees' )
FROM dummy;
```

SQLFLAGGER 関数 [その他]

指定した規格に対する、指定した SQL 文の準拠性を返します。

構文

SQLFLAGGER(*sql-standard-string*, *sql-statement-string* **)**

パラメーター

- ***sql-standard-string*** 準拠をテストする規格レベル。使用できる値は、sql_flagger_error_level データベースオプションと同じです。
  - **SQL:2008/Core** コア SQL/2008 構文に対する準拠性をテストします。
  - **SQL:2008/Package** 上級レベルの SQL/2008 構文に対する準拠性をテストします。
  - **SQL:2003/Core** コア SQL/2003 構文に対する準拠性をテストします。
  - **SQL:2003/Package** 上級レベルの SQL/2003 構文に対する準拠性をテストします。

- ○ **SQL:1999/Core**　コア SQL/1999 構文に対する準拠性をテストします。
- ○ **SQL:1999/Package**　上級レベルの SQL/1999 構文に対する準拠性をテストします。
- ○ **SQL:1992/Entry**　初級レベルの SQL/1992 構文に対する準拠性をテストします。
- ○ **SQL:1992/Intermediate**　中級レベルの SQL/1992 構文に対する準拠性をテストします。
- ○ **SQL:1992/Full**　上級レベルの SQL/1992 構文に対する準拠性をテストします。
- ○ **UltraLite**　Ultra Light に対する準拠性をテストします。

- **sql-statement-string**　準拠性をチェックする SQL 文。

戻り値

LONG VARCHAR

参照

- 「sql_flagger_error_level オプション」『SQL Anywhere サーバー データベース管理』
- 「SQL プリプロセッサー」『SQL Anywhere サーバー プログラミング』
- 「sa_ansi_standard_packages システムプロシージャー」1042 ページ
- 「SQL Flagger を使用した SQL 準拠のテスト」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、使用できない拡張機能が検出されたときに返されるメッセージの例を示します。

```
SELECT SQLFLAGGER(
  'SQL:2003/Package', 'SELECT top 1 dummy_col FROM sys.dummy ORDER BY dummy_col' );
```

この文は、メッセージ '0AW03 行 1 の 'top' の付近に、言語の使用できない拡張機能が検出されました。' を返します。

次の文は、使用できない拡張機能を含んでいないため、'00000' を返します。

```
SELECT SQLFLAGGER( 'SQL:2003/Package', 'SELECT dummy_col FROM sys.dummy' );
```

SQRT 関数 [数値]

数の平方根を返します。

構文

SQRT(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　平方根が計算される数。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

標準と互換性

- **SQL/2008**　SQRT 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。

例

次の文は、値 3 を返します。

```
SELECT SQRT( 9 );
```

STDDEV 関数 [集合]

STDDEV_SAMP のエイリアスです。

参照

- 「STDDEV_SAMP 関数 [集合]」356 ページ

STDDEV_POP 関数 [集合]

数値式からなる母集団の標準偏差を DOUBLE として計算します。

構文 1

STDDEV_POP(*numeric-expression* **)**

構文 2

STDDEV_POP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***numeric-expression***　ローセットで母集団ベースの標準偏差を計算する対象の式。通常、式はカラム名です。

戻り値

DOUBLE

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

母集団ベースの標準偏差は、次の式によって計算されます。

$s = [(1/N) * SUM(x_I - mean(x))^2]^{½}$

この標準偏差には、*numeric-expression* が NULL 値のローは含まれません。グループにローが含まれていない場合は、NULL を返します。

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008**　STDDEV_POP 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。Window 関数として使用する場合、STDDEV_POP はオプションの SQL 基本機能 T611、「Elementary OLAP operations」の一部です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、STDDEV_POP 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の文は、異なる期間における注文ごとの項目数で平均と平方偏差をリストします。

```
SELECT YEAR( ShipDate ) AS Year,
   QUARTER( ShipDate ) AS Quarter,
```

```
    AVG( Quantity ) AS Average,
    STDDEV_POP( quantity ) AS Variance
FROM 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...
...	...	...	...

STDDEV_SAMP 関数 [集合]

数値式からなるサンプルの標準偏差を DOUBLE として計算します。

構文 1

STDDEV_SAMP(*numeric-expression* **)**

構文 2

STDDEV_SAMP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***numeric-expression*** ローセットでサンプルベースの標準偏差を計算する対象の式。通常、式はカラム名です。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、倍精度浮動小数点で計算を行います。

標準偏差は、次の式によって計算されます。これは、正規分布を前提としています。

$s = [(1/(N - 1)) * SUM(x_I - mean(x))^2]^{½}$

この標準偏差には、*numeric-expression* が NULL 値のローは含まれません。グループに 0 か 1 のローが含まれている場合は、NULL を返します。

実行される統計計算の詳細については、「集合関数に対応する数式」『SQL Anywhere サーバー SQL の使用法』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「集合関数」139 ページ

標準と互換性

- **SQL/2008** STDDEV_SAMP 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。Window 関数として使用する場合、STDDEV_SAMP はオプションの SQL 基本機能 T611、「Elementary OLAP operations」の一部です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、STDDEV_SAMP 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の文は、異なる期間における注文ごとの項目数で平均と平方偏差をリストします。

```
SELECT YEAR( ShipDate ) AS Year,
   QUARTER( ShipDate ) AS Quarter,
   AVG( Quantity ) AS Average,
   STDDEV_SAMP( quantity ) AS Variance
FROM 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...
...	...	...	...

STR 関数 [文字列]

指定した数に相当する文字列を返します。

構文

STR(*numeric-expression* [, *length* [, *decimal*]] **)**

パラメーター

- ***numeric-expression*** -1E126 と 1E127 との間の任意の概数 (浮動小数点、実数、または倍精度)。
- ***length*** 返される文字数 (小数点、小数点の左右のすべての桁、ブランクを含む)。デフォルトは 10 です。
- ***decimal*** 返される小数点以下の桁数。デフォルトは 0 です。

戻り値

VARCHAR

備考

数の整数部分が指定した長さに合わない場合は、指定した長さの文字列がすべてアスタリスクで埋められて返されます。たとえば、次の文は *** を返します。

```
SELECT STR( 1234.56, 3 );
```

> **注意**
> サポートされている最大長は 128 です。長さが 1 ～ 128 の範囲外である場合は、結果が NULL になります。

参照

- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、6 個のスペースと 1235 (合計 10 文字) の文字列を返します。

```
SELECT STR( 1234.56 );
```

次の文は、結果 1234.6 を返します。

```
SELECT STR( 1234.56, 6, 1 );
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

STRING 関数 [文字列]

1 つ以上の文字列を連結して 1 つの長い文字列にします。

構文

STRING(*string-expression* **[, ...])**

パラメーター

- ***string-expression*** 評価される文字列。

  引数を 1 つだけ指定する場合は、1 つの式に変換されます。複数の引数を指定する場合は、連結されて 1 つの文字列になります。

戻り値

- LONG VARCHAR
- LONG NVARCHAR
- LONG BINARY

備考

数値または日付をパラメーターとして指定した場合は、文字列に変換されてから連結されます。また、1 つの式を唯一のパラメーターとして指定すると、STRING 関数を使用して、その式を文字列に変換できます。

すべてのパラメーターが NULL の場合、STRING は NULL を返します。NULL でないパラメーターが存在すると、NULL パラメーターはすべて空の文字列として処理されます。

参照

- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 *testing123* を返します。

```
SELECT STRING( 'testing', NULL, 123 );
```

STRTOUUID 関数 [文字列]

文字列の値をユニークな識別子 (UUID または GUID) の値に変換します。

注意
バージョン 9.0.2 より前に作成されたデータベースでは、UNIQUEIDENTIFIER データ型はユーザー定義データ型として定義され、UUID 値のバイナリ表現と文字列表現の間を変換するための STRTOUUID 関数と UUIDTOSTR 関数が必要です。

バージョン 9.0.2 以降を使用して作成されたデータベースでは、UNIQUEIDENTIFIER データ型がネイティブデータ型に変更されています。データ型の変換は、SQL Anywhere が必要に応じて実行します。これらのバージョンでは STRTOUUID 関数と UUIDTOSTR 関数を使用する必要はありません。

詳細については、「UNIQUEIDENTIFIER データ型」123 ページを参照してください。

構文

STRTOUUID(*string-expression* **)**

パラメーター

- ***string-expression*** フォーマットが *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* の文字列。

戻り値

UNIQUEIDENTIFIER

備考

フォーマットが *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* の文字列をユニークな識別子の値に変換します。ここで、*x* は 16 進数です。

文字列が有効な UUID 文字列でない場合は、conversion_error オプションが OFF に設定されていないかぎり変換エラーが返されます。このオプションが OFF の場合は NULL が返されます。

この関数は、UUID 値をデータベースに挿入するときに役立ちます。

この関数は NCHAR の入力または出力をサポートしています。

中カッコは、*string-expression* の最初と最後の文字として使用できます。

参照

- 「UUIDTOSTR 関数 [文字列]」383 ページ
- 「NEWID 関数 [その他]」286 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は同義であり、結果 0x6c2b64a93c6f47dc901536b9ed49fec2 を返します。

```
SELECT STRTOUUID( '6c2b64a9-3c6f-47dc-9015-36b9ed49fec2' );
SELECT STRTOUUID( '{6c2b64a9-3c6f-47dc-9015-36b9ed49fec2}' );
```

STUFF 関数 [文字列]

1 つの文字列から複数の文字を削除して、別の文字列に置き換えます。

構文

STUFF(*string-expression-1*, *start*, *length*, *string-expression-2* **)**

パラメーター

- ***string-expression-1*** STUFF 関数によって変更される文字列。
- ***start*** 削除を開始する文字の位置。文字列の先頭文字の位置を 1 とします。
- ***length*** 削除する文字数。
- ***string-expression-2*** 挿入する文字列。STUFF 関数を使用して文字列の一部を削除するには、NULL の置換文字列を使用します。

戻り値

LONG NVARCHAR

備考

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「INSERTSTR 関数 [文字列]」256 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 chocolate pie を返します。

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

SUBSTRING 関数 [文字列]

文字列の部分文字列を返します。

構文

{ **SUBSTRING** | **SUBSTR** } **(** *string-expression*, *start*
[, *length*] **)**

パラメーター

- ***string-expression*** 部分文字列が返される文字列。

- ***start*** 文字単位で指定した、返される部分文字列の開始位置。
- ***length*** 文字単位で指定した、返される部分文字列の長さ。*length* を指定すると、部分文字列は指定した長さに限定されます。

戻り値

- LONG BINARY
- LONG VARCHAR
- LONG NVARCHAR

備考

この関数の動作は、ansi_substring データベースオプションの設定によって変わります。ansi_substring オプションを On (デフォルト) に設定した場合、SUBSTRING 関数は ANSI/ISO SQL/2008 と同じ動作をします。動作を次に示します。

ansi_substring オプション設定	start 値	length 値
On	文字列の最初の文字の位置を 1 とします。負または 0 の開始オフセットは、文字列の左側が文字以外で埋められているように扱われます。	正の *length* は、部分文字列が開始位置の右側から *length* 文字で終わることを示します。 負の *length* はエラーを返します。
Off	文字列の先頭文字の位置を 1 とします。負の開始位置は、文字列の先頭からではなく、末尾からの文字数を指定します。 *start* が 0 で length が負でない場合は、start 値として 1 が使用されます。start が 0 で *length* が負の場合は、start 値として -1 が使用されます。	正の *length* は、部分文字列が開始位置の右側から *length* 文字で終わることを示します。 負の *length* は、開始位置から最大 *length* 文字左側の文字を返します。

string-expression が binary データ型の場合、SUBSTRING 関数は BYTE_SUBSTR のように動作します。

文字列の末尾の文字を取得するには、RIGHT 関数を使用します。

この関数は NCHAR の入力または出力をサポートしています。入力文字が文字長のセマンティックを使用している場合、可能であれば、戻り値が文字長のセマンティックで記述されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「BYTE_SUBSTR 関数 [文字列]」 164 ページ
- 「LEFT 関数 [文字列]」 265 ページ
- 「RIGHT 関数 [文字列]」 334 ページ
- 「CHARINDEX 関数 [文字列]」 169 ページ
- 「文字列関数」 148 ページ

標準と互換性

- **SQL/2008**　SUBSTRING は、SQL/2008 標準のコア機能です。標準の実装内容は SQL Anywhere の実装内容とはわずかに異なります。標準ではキーワード FROM と FOR を使用して SUBSTRING を 3 つのパラメーターで定義しますが、SQL Anywhere ではこれらのキーワードはいずれも必要ありません。

例

次の表は、SUBSTRING 関数から返される値を示します。

例	結果
SUBSTRING('front yard', 1, 4)	fron
SUBSTRING('back yard', 6, 4)	yard
SUBSTR('abcdefgh', 0, -2)	ansi_substring が On の場合、エラーを返します
SUBSTR('abcdefgh', -2, 2)	ansi_substring が On の場合、空の文字列を返します

SUM 関数 [集合]

ローグループごとに、指定された式の合計を返します。

構文 1

SUM([**ALL** | **DISTINCT**] *expression* **)**

構文 2

SUM([**ALL**] *expression* **) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- **[ALL] *expression***　合計される式の名前。通常はカラム名です。
- **DISTINCT *expression***　各グループの *expression* で一意の値の合計を計算します。

戻り値

- INTEGER
- DOUBLE
- NUMERIC

備考

指定された式が NULL のローは含まれません。

グループにローが含まれていない場合は、NULL を返します。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

この関数ではオーバーフローエラーが発生することがあり、これにより、エラーが返される場合があります。オーバーフローエラーを回避するために、*numeric-expression* で CAST 関数を使用できます。「CAST 関数 [データ型変換]」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「WINDOW 句」989 ページ
- 「COUNT 関数 [集合]」183 ページ
- 「AVG 関数 [集合]」156 ページ

標準と互換性

- **SQL/2008** コア機能。Window 関数として使用する場合 (構文 2)、SUM はオプションの SQL/2008 言語機能 T611、「Elementary OLAP operations」の一部です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、SUM 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。「集合関数と外部参照」『SQL Anywhere サーバー SQL の使用法』を参照してください。

  例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、値 3749146.740 を返します。

```
SELECT SUM( Salary )
FROM Employees;
```

SUSER_ID 関数 [システム]

指定したユーザー名の数値のユーザー ID を返します。

構文

SUSER_ID([*user-name*])

パラメーター

- ***user-name*** 検索しているユーザー ID のユーザー名。

戻り値

INT

備考

user-name を指定しない場合は、現在のユーザーの ID が返されます。

参照

- 「SUSER_NAME 関数 [システム]」365 ページ
- 「USER_ID 関数 [システム]」381 ページ

標準と互換性

- **SQL/2008** Transact-SQL 拡張。

例

次の文は、GROUPO ユーザーの ID を返します。

```
SELECT SUSER_ID( 'GROUPO' );
```

SUSER_NAME 関数 [システム]

指定したユーザー ID のユーザー名を返します。

構文

SUSER_NAME([*user-id*])

パラメーター

- ***user-id*** 検索しているユーザーのユーザー ID。

戻り値

LONG VARCHAR

備考

user-id を指定しない場合は、現在のユーザーのユーザー名が返されます。

参照

- 「SUSER_ID 関数 [システム]」365 ページ
- 「USER_NAME 関数 [システム]」382 ページ

標準と互換性

- **SQL/2008** Transact-SQL 拡張。

例

次の文は、ID 101 のユーザーのユーザー名を返します。

```
SELECT SUSER_NAME( 101 );
```

SWITCHOFFSET 関数 [日付と時刻]

元のタイムゾーンオフセットから指定のタイムゾーンオフセットに変換される TIMESTAMP WITH TIME ZONE 値を返します。

構文

SWITCHOFFSET(*tmz-expression*, *time-zone-offset* **)**

パラメーター

- ***tmz-expression*** 変換される TIMESTAMP WITH TIME ZONE 値。
- ***time-zone-offset*** 結果のタイムゾーンオフセット。協定世界時 (UTC: Coordinated Universal Time) の前または後の分数を表す整数、{ + | - } hh:nn 形式の文字列、またはズールータイムゾーンを表す Z のいずれかをこの値として使用できます。ズールータイムゾーンは、UTC と同じタイムゾーンです。

戻り値

TIMESTAMP WITH TIME ZONE

参照

- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「SYSDATETIMEOFFSET 関数 [日付と時刻]」367 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、タイムゾーンオフセット値を -04:00 時から -07:00 時に変更します。戻り値は 2009-04-03 11:45:12.123-07:00 です。

```
SELECT CAST ( '2009-04-03 14:45:12.123-04:00' AS datetimeoffset ) AS EDT,
SWITCHOFFSET( EDT,'-07:00' ) AS PDT;
```

SYSDATETIMEOFFSET 関数 [日付と時刻]

システムクロックを使用するデータベースサーバーの現在の日付、時刻、およびタイムゾーンオフセットを返します。

構文

SYSDATETIMEOFFSET ( )

戻り値

TIMESTAMP WITH TIME ZONE

参照

- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ
- 「SWITCHOFFSET 関数 [日付と時刻]」366 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、データベースサーバーの現在の日付と時刻およびタイムゾーンオフセットを返します。

```
SELECT SYSDATETIMEOFFSET ( );
```

次の例は、SYSDATETIMEOFFSET 値をクライアントコンピューターのタイムゾーンに変換します。

```
SELECT SWITCHOFFSET ( SYSDATETIMEOFFSET ( ),
CAST( connection_property ('TimeZoneAdjustment') AS INT ));
```

TAN 関数 [数値]

数のタンジェントを返します。

構文

TAN(*numeric-expression* **)**

パラメーター

- ***numeric-expression***　角度 (ラジアン)。

戻り値

DOUBLE

備考

ATAN 関数と TAN 関数は逆変換の演算です。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「COS 関数 [数値]」182 ページ
- 「SIN 関数 [数値]」345 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、0.52 の tan 値を返します。

```
SELECT TAN( 0.52 );
```

TEXTPTR 関数 [テキストとイメージ]

指定したカラムへの 16 バイトのバイナリポインターを返します。この機能は Transact-SQL との互換性のためにのみ提供されているものであり、使用しないことをおすすめします。

構文

TEXTPTR(*column-name* **)**

パラメーター

- ***column-name*** CHAR、NCHAR、または BINARY のデータが含まれるカラムの名前。

戻り値

BINARY

備考

この関数は、Transact-SQL との互換性を保つために実装されています。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の Embedded SQL の例では、TEXTPTR を使用して、MarketingInformation テーブルの ProductID 500 に関連付けられている Description カラムを見つけます。

テキストポインターは、変数 **txtptr** に格納され、READTEXT 文のパラメーターとして指定されます。READTEXT 文は、カラムオフセット 181 から開始して 55 バイトを返します。

```
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 は次の文字列を返します。

```
Lightweight 100% organically grown cotton construction.
```

TO_CHAR 関数 [文字列]

任意のサポートされている文字セットの文字データを、データベースの CHAR 文字セットに変換します。

構文

TO_CHAR(*string-expression* **[,** *source-charset-name* **])**

パラメーター

- ***string-expression*** 変換される文字列。
- ***source-charset-name*** 文字列の文字セット。

戻り値

LONG VARCHAR

備考

source-charset-name を指定すると、この関数は次と同等になります。

```
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
 'db_charset', source-charset-name )
   AS CHAR );
```

db_charset の詳細については、「CSCONVERT 関数 [文字列]」189 ページを参照してください。

source-charset-name を指定しないと、この関数は次と同等になります。

```
CAST( string-expression AS CHAR );
```

参照

- 「推奨文字セットと照合」『SQL Anywhere サーバー データベース管理』
- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」175 ページ
- 「CSCONVERT 関数 [文字列]」189 ページ
- 「NCHAR 関数 [文字列]」286 ページ
- 「TO_NCHAR 関数 [文字列]」370 ページ
- 「UNICODE 関数 [文字列]」379 ページ
- 「UNISTR 関数 [文字列]」380 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

cp850 文字セットを含む BINARY 値の場合、次の文はデータを CHAR の文字セットとデータ型に変換します。

```
SELECT TO_CHAR( 'cp850_data', 'cp850' );
```

TO_NCHAR 関数 [文字列]

任意のサポートされている文字セットの文字データを、NCHAR 文字セットに変換します。

構文

TO_NCHAR(*string-expression* **[,** *source-charset-name* **])**

パラメーター

- ***string-expression***　変換される文字列。
- ***source-charset-name***　文字列の文字セット。

戻り値

LONG NVARCHAR

備考

source-charset-name を指定すると、この関数は次と同等になります。

```
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
 'nchar_charset', source-charset-name )
 AS NCHAR );
```

nchar_charset の詳細については、「CSCONVERT 関数 [文字列]」189 ページを参照してください。

source-charset-name を指定しないと、この関数は次と同等になります。

```
CAST( string-expression AS NCHAR );
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「推奨文字セットと照合」『SQL Anywhere サーバー データベース管理』
- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」175 ページ
- 「CSCONVERT 関数 [文字列]」189 ページ
- 「NCHAR 関数 [文字列]」286 ページ
- 「TO_CHAR 関数 [文字列]」369 ページ
- 「UNICODE 関数 [文字列]」379 ページ
- 「UNISTR 関数 [文字列]」380 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

cp850 文字セットを含む BINARY 値の場合、次の例はデータを NCHAR の文字セットとデータ型に変換します。

```
SELECT TO_NCHAR( 'cp850_data', 'cp850' );
```

TODATETIMEOFFSET 関数 [日付と時刻]

指定のタイムゾーンオフセットを使用して、TIMESTAMP 値を TIME STAMP WITH TIME ZONE 値に変換します。

構文

TODATETIMEOFFSET(*timestamp-expression*, *time-zone-offset* **)**

パラメーター

- ***timestamp-expression***　変換される TIMESTAMP 式。

- ***time-zone-offset***　タイムゾーンオフセット。UTC の前または後の分数を表す INTEGER、{ + | - } hh:nn 形式の VARCHAR、またはズールータイムゾーンを表す文字列 "Z" のいずれかをこの値として使用できます。ズールータイムゾーンは、UTC と同じタイムゾーンです。

戻り値

TIMESTAMP WITH TIME ZONE

参照

- 「TIMESTAMP WITH TIME ZONE データ型」120 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、TIMESTAMP 値を TIMESTAMP WITH TIME ZONE 値に変換します。

```
SELECT CAST('2009-04-03 14:45:12.123' AS TIMESTAMP) AS orig,
    TODATETIMEOFFSET (orig,'+11:00');
```

TODAY 関数 [日付と時刻]

現在の日付を DATE 値で返します。

構文

TODAY([*])

戻り値

DATE

備考

セマンティック上、TODAY(*) と TODAY() は同義です。TODAY は CURRENT DATE 特別値と同義です。

要求に含まれる TODAY 関数の各インスタンスは、多くても 1 回のみ評価されます。同じ要求に含まれる TODAY の複数のインスタンスは、同じ DATE 値を共有する場合もあれば、そうでない場合もあります。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、システムクロックによる現在の日付を返します。

```
SELECT TODAY( * );
SELECT CURRENT DATE;
```

TRACEBACK 関数 [その他]

最近の例外 (エラー) の発生時に実行中であったプロシージャーやトリガーのトレースバックが含まれた文字列を返します。

構文

TRACEBACK([*])

戻り値

LONG VARCHAR

備考

セマンティック上、TRACEBACK(*) と TRACEBACK() は同義です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

この関数は、プロシージャーとトリガー (特に、Transact-SQL ダイアレクトで記述されている場合) のデバッグに役立ちます。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

traceback 関数を使用するには、プロシージャーの実行中にエラーが発生した後で次のように入力します。

```
SELECT TRACEBACK( * );
```

TRACED_PLAN 関数 [その他]

この関数は、Sybase Central でトレーシングデータを使用してクエリのグラフィカルなプランを生成するときに使用します。

構文

TRACED_PLAN(*logging_session_id*, *query_id* **)**

パラメーター

- ***logging_session_id*** この INTEGER パラメーターと *query_id* を組み合わせると、プランを生成する sa_diagnostic_query のローを識別できます。
- ***query_id*** この INTEGER パラメーターと *logging_session_id* を組み合わせると、プランを生成する sa_diagnostic_query のローを識別できます。

戻り値

LONG VARCHAR

備考

この関数は Sybase Central から使用されます。

参照

- 「sa_diagnostic_query テーブル」1017 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

TRANSACTSQL 関数 [その他]

Watcom SQL 文を取得し、Transact-SQL ダイアレクトで書き換えます。

構文

TRANSACTSQL(*sql-statement-string* **)**

パラメーター

- ***sql-statement-string*** Transact-SQL 表現に書き換えられる SQL 文。

戻り値

LONG VARCHAR

参照

- 「SQLDIALECT 関数 [その他]」352 ページ
- 「WATCOMSQL 関数 [その他]」388 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、文字列 'SELECT EmployeeName=empl_name FROM Employees' を返します。

```
SELECT TRANSACTSQL( 'SELECT empl_name as EmployeeName FROM Employees' ) FROM
dummy;
```

TREAT 関数 [データ型変換]

TREAT 関数では、ジオメトリ式の宣言されたタイプをサブタイプに変更できます。この関数は、空間データで使用します。

構文

TREAT (*geometry-expression* AS *subtype* **)**

パラメーター

- ***geometry-expression*** 変換される式。
- ***subtype*** *geometry-expression* の変換後のサブタイプ。

戻り値

要求されたデータ型によって異なります。

備考

TREAT 関数は、ジオメトリでのみ使用できます。

式の動的タイプがターゲットのデータ型のサブタイプでない場合は、エラーが返されます。また、CAST 関数を使用して、ジオメトリ式が宣言されたタイプを変更することもできます。ただし、CAST 関数では、サブタイプ階層の外側を変更できます。たとえば、CAST を使用して、ポイントをマルチポイントに変換することができます。このような変換を実行すると、式の動的タ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

イプに予期しない変更が加えられることがあるため、スーパータイプからサブタイプに移動する場合は TREAT 関数を使用することをおすすめします。また、TREAT 関数は、CAST 関数と比較して、より効率的に実行されます。

参照

- 「サブタイプへの TREAT 式の使用」『SQL Anywhere サーバー 空間データサポート』
- 「CAST 関数 [データ型変換]」 165 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

Interactive SQL で次の文を実行して、テーブルを作成し、2 つの値をロードします。

```
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 ) ) );
```

次のクエリはエラーを返します。「タイプ '%1' には '%2' というメソッドはありません ('%3' 付近)。」『エラーメッセージ』を参照してください。

```
SELECT geo.ST_X() FROM treatExample T WHERE pk = 0;
```

次のクエリは成功します。

```
SELECT TREAT( geo AS ST_Point ).ST_X() FROM treatExample WHERE pk = 0;
```

次のクエリはエラーを返します。「タイプ '%1' には '%2' というメソッドはありません ('%3' 付近)。」『エラーメッセージ』を参照してください。

```
SELECT TREAT( geo AS ST_Point ).ST_X() FROM treatExample T WHERE pk = 0;
```

次の例では、TREAT 文の代わりに CAST 文が使用されているため、成功します。

```
SELECT CAST( geo AS ST_Point ) FROM treatExample WHERE pk = 1;
```

TRIM 関数 [文字列]

先行ブランクと後続ブランクを文字列から削除します。

構文

TRIM(*string-expression* **)**

パラメーター

- ***string-expression*** 削除する文字列。

戻り値

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

備考

この関数は NCHAR の入力または出力をサポートしています。

参照

- 「LTRIM 関数 [文字列]」274 ページ
- 「RTRIM 関数 [文字列]」338 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　TRIM 関数は SQL/2008 のコア機能です。

  SQL Anywhere では、SQL/2008 で定義されている追加のパラメーター *trim specification* と *trim character* をサポートしていません。SQL Anywhere の TRIM の実装は、BOTH の TRIM 仕様に対応しています。

  SQL/2008 標準で定義される他の TRIM 仕様 (LEADING と TRAILING) については、SQL Anywhere で LTRIM 関数と RTRIM 関数がそれぞれ提供されています。

例

次の文は、先行ブランクと後続ブランクなしの値 chocolate を返します。

```
SELECT TRIM( '  chocolate  ' );
```

TRUNCNUM 関数 [数値]

指定した桁数で小数点以下を切り捨てます。

構文

{ **TRUNCNUM** | **TRUNCATE** }(*numeric-expression*, *integer-expression*)

パラメーター

- ***numeric-expression***　トランケートされる数。この引数には、NUMERIC または DOUBLE データ型を使用できます。

- ***integer-expression***　正の整数は、丸めを行う小数点の右側の有効桁数を指定します。負の値は、丸めを行う小数点の左側の有効桁数を指定します。

戻り値

NUMERIC または DOUBLE

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

数字をトランケートする場合、TRUNCATE 関数ではなく TRUNCNUM を使用します。

TRUNCATE 関数の使用はおすすめしません。truncate という単語はキーワードなので、quoted_identifier オプションを OFF に設定するか、単語 TRUNCATE を引用符で囲む必要があるためです。

参照

- 「ROUND 関数 [数値]」335 ページ
- 「quoted_identifier オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 600 を返します。

```
SELECT TRUNCNUM( 655, -2 );
```

次の文は、値 655.340 を返します。

```
SELECT TRUNCNUM( 655.348, 2 );
```

TSEQUAL 関数 [システム] (旧式)

2 つの TIMESTAMP 値を比較し、これらが同じかどうかを返します。

構文

TSEQUAL (*timestamp-expression-1*, *timestamp-expression-2* **)**

パラメーター

- ***timestamp-expression-1***　TIMESTAMP 値。
- ***timestamp-expression-2***　TIMESTAMP 値。

戻り値

BIT

備考

TSEQUAL 関数は、WHERE 句でのみ使用できます。これは、UPDATE 文の一部として最も一般的に使用されます。

TSEQUAL 関数は通常の 2 つの TIMESTAMP 値の比較に使用できますが、TSEQUAL の目的は、2 つの特殊な Transact-SQL TIMESTAMP 値を比較することでローが他の接続によって変更されたかどうかを判断することにあります。

TSEQUAL を使用するシングルロー の UPDATE 文で、*timestamp-expression-1* と *timestamp-expression-2* が等しく、このうちの一方が DEFAULT TIMESTAMP で宣言されたカラムを参照しており、ローが最後にフェッチされたときのカラム値をもう一方が参照している場合は、フェッチ後にローが変更されていないため、TSEQUAL は TRUE を返します。別のユーザーがローを変更した場合は、そのタイプスタンプが変更されており、TSEQUAL 関数は FALSE を返します。TSEQUAL 関数がこの場合に FALSE を返すと、UPDATE は実行されません。アプリケーションでは、影響を受けるロー数を @@rowcount などを使用して調べることにより、ローが更新されたかどうかを判断できます。影響を受けるローがない場合は、別のユーザーがローを変更したと想定でき、再フェッチが必要になります。

参照

- 「TIMESTAMP カラムのデータ型」『SQL Anywhere サーバー SQL の使用法』
- 「TIMESTAMP 特別値」73 ページ
- 「Transact-SQL の特殊な TIMESTAMP カラムとデータ型」『SQL Anywhere サーバー SQL の使用法』
- 「UPDATE 文」976 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

TIMESTAMP カラム Products.LastUpdated を作成し、ローが最後に更新された時刻のタイムスタンプを格納するとします。次の例は、TSEQUAL 関数を使用してローの値を変更します。最後にフェッチされた後でローに変更がない場合にのみ、更新が適用されます。

```
SELECT LastUpdated into old_ts_value
FROM Products
WHERE ID = '300';

UPDATE Products
SET Color = 'Yellow'
WHERE ID = '300'
AND TSEQUAL( LastUpdated, old_ts_value );
```

UCASE 関数 [文字列]

文字列中のすべての文字を大文字に変換します。

構文

UCASE(*string-expression* **)**

パラメーター

- ***string-expression***　大文字に変換される文字列。

戻り値

引数のデータ型に対応する CHAR、VARCHAR、LONG VARCHAR、NCHAR、NVARCHAR、LONG NVARCHAR。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

この関数は UPPER 関数と同じです。

参照

- 「UPPER 関数 [文字列]」381 ページ
- 「LCASE 関数 [文字列]」264 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。UPPER 関数は SQL/2008 に準拠しています。

例

次の文は、値 CHOCOLATE を返します。

```
SELECT UCASE( 'ChocoLate' );
```

UNICODE 関数 [文字列]

文字列の最初の文字に Unicode のコードポイントを含む整数を返します。最初の文字が有効なエンコーディングではない場合は NULL を返します。

構文

UNICODE(*nchar-string-expression* **)**

パラメーター

- ***nchar-string-expression***　最初の文字が整数に変換される NCHAR 文字列。

戻り値

INT

参照

- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」175 ページ
- 「NCHAR 関数 [文字列]」286 ページ
- 「TO_CHAR 関数 [文字列]」369 ページ
- 「TO_NCHAR 関数 [文字列]」370 ページ
- 「UNISTR 関数 [文字列]」380 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、整数 65536 を返します。

```
SELECT UNICODE(UNISTR( '¥u010000data' ));
```

UNISTR 関数 [文字列]

文字と Unicode エスケープシーケンスで構成される文字列を NCHAR 文字列に変換します。

構文

UNISTR(*string-expression* **)**

パラメーター

- ***string-expression*** 変換される文字列。

戻り値

- NVARCHAR
- LONG NVARCHAR

備考

UNISTR 関数を使用すると、SQL 文で使用される CHAR 文字セットで表現できない Unicode 文字を使用できるようになります。たとえば、英語環境では UNISTR 関数を使用すると、漢字を使用できるようになります。

UNISTR 関数には N" 定数と機能が似ています。ただし、UNISTR 関数は Unicode 文字と CHAR 文字セットの文字を使用できますが、N" 定数は CHAR 文字セットの文字のみを使用できます。

string-expression には文字と Unicode のエスケープシーケンスが含まれます。Unicode のエスケープシーケンス形式は、¥uXXXX または ¥uXXXXXX です (各 X は 16 進数)。UNISTR 関数は、個々の文字と Unicode のエスケープシーケンスを対応する Unicode 文字に変換します。

6 桁の Unicode のエスケープシーケンスを使用する場合、値は Unicode のコードポイントの最大値である 10FFFF を超えません。¥u234567 などのシーケンスは 6 桁の Unicode エスケープシーケンスではありません。4 桁のシーケンス ¥u2345 の後に、文字 6 と 7 が続きます。

2 つの隣接する Unicode エスケープシーケンスが UTF-16 の代理ペアを構成する場合、出力時に 1 つの Unicode 文字に結合されます。

参照

- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」175 ページ
- 「NCHAR 関数 [文字列]」286 ページ
- 「TO_CHAR 関数 [文字列]」369 ページ
- 「TO_NCHAR 関数 [文字列]」370 ページ
- 「UNICODE 関数 [文字列]」379 ページ
- 「文字列」5 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、文字列 Hello を返します。

```
SELECT UNISTR( 'Hel¥u006c¥u006F' );
```

次の例は、UTF-16 の代理ペア D800-DF02 を Unicode コードポイント 10302 に結合します。

```
SELECT UNISTR( '¥uD800¥uDF02' );
```

この例は前の例と同等です。

```
SELECT UNISTR( '¥u010302' );
```

UPPER 関数 [文字列]

文字列中のすべての文字を大文字に変換します。

構文

UPPER(*string-expression* **)**

パラメーター

- ***string-expression*** 大文字に変換される文字列。

戻り値

引数のデータ型に対応する CHAR、VARCHAR、LONG VARCHAR、NCHAR、NVARCHAR、LONG NVARCHAR。

備考

この関数は UCASE 関数と同じです。

参照

- 「UCASE 関数 [文字列]」378 ページ
- 「LCASE 関数 [文字列]」264 ページ
- 「LOWER 関数 [文字列]」273 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** UPPER 関数は、SQL/2008 標準のコア機能です。

例

次の文は、値 CHOCOLATE を返します。

```
SELECT UPPER( 'ChocoLate' );
```

USER_ID 関数 [システム]

指定したユーザー名の数値のユーザー ID を返します。

構文

USER_ID([*user-name*] **)**

パラメーター

- ***user-name*** 検索しているユーザー ID のユーザー名。

戻り値

INTEGER

備考

user-name を指定しない場合は、現在のユーザーの ID が返されます。

参照

- 「USER_NAME 関数 [システム]」382 ページ
- 「SUSER_ID 関数 [システム]」365 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、GROUPO ユーザーの ID を返します。

```
SELECT USER_ID( 'GROUPO' );
```

USER_NAME 関数 [システム]

指定したユーザー ID のユーザー名を返します。

構文

USER_NAME([*user-id*] **)**

パラメーター

- ***user-id*** 検索しているユーザーのユーザー ID。

戻り値

LONG VARCHAR

備考

user-id を指定しない場合は、現在のユーザーのユーザー名が返されます。

参照

- 「USER_ID 関数 [システム]」381 ページ
- 「SUSER_NAME 関数 [システム]」365 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、ユーザー ID 101 のユーザー名を返します。

```
SELECT USER_NAME( 101 );
```

UUIDTOSTR 関数 [文字列]

ユニークな識別子の値 (UUID または GUID) を文字列の値に変換します。

> **注意**
> バージョン 9.0.2 より前に作成されたデータベースでは、UNIQUEIDENTIFIER データ型はユーザー定義データ型として定義され、UUID 値のバイナリ表現と文字列表現の間を変換するための STRTOUUID 関数と UUIDTOSTR 関数が必要です。
>
> バージョン 9.0.2 以降を使用して作成されたデータベースでは、UNIQUEIDENTIFIER データ型がネイティブデータ型に変更されています。データ型の変換は、SQL Anywhere が必要に応じて実行します。これらのバージョンでは STRTOUUID 関数と UUIDTOSTR 関数を使用する必要はありません。
>
> 詳細については、「UNIQUEIDENTIFIER データ型」123 ページを参照してください。

構文

UUIDTOSTR(*uuid-expression* **)**

パラメーター

- ***uuid-expression***　ユニークな識別子の値。

戻り値

VARCHAR

備考

ユニークな識別子を、フォーマットが *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* の文字列値に変換します。ここで、x は 16 進数です。バイナリ値が有効な uniqueidentifier でない場合は、NULL を返します。

この関数は、UUID 値を表示する場合に役立ちます。

参照

- 「NEWID 関数 [その他]」286 ページ
- 「STRTOUUID 関数 [文字列]」359 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、2 つのカラムを持つテーブル mytab を作成します。カラム pk は uniqueidentifier データ型で、カラム c1 は integer データ型です。それぞれに値 1 と値 2 を持つ 2 つのローをカラム c1 に挿入します。

```
CREATE TABLE mytab(
   pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
   c1 INT );
INSERT INTO mytab( c1 ) values ( 1 );
INSERT INTO mytab( c1 ) values ( 2 );
```

次の SELECT 文を実行すると、新規に作成したテーブルのすべてのデータを返します。

```
SELECT * FROM mytab;
```

2 つのカラムと 2 つのローを持ったテーブルが表示されます。カラム pk に表示される値は、バイナリ値です。

ユニークな識別子の値を読みやすい形式に変換するには、次のコマンドを実行します。

```
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

UUIDTOSTR 関数は、バージョン 9.0.2 以降で作成されたデータベースでは不要です。

VAR_POP 関数 [集合]

数値式からなる母集団の統計上の平方偏差を DOUBLE として計算します。

構文 1

VAR_POP(*numeric-expression* **)**

構文 2

VAR_POP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***numeric-expression*** ローセットで母集団ベースの平方偏差を計算する対象の式。通常、式はカラム名です。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

numeric-expression (x) の母集団ベースの平方偏差 (s^2) は、次の式によって計算されます。

$s^2 = (1/N) * SUM(x_i - mean(x))^2$

この平方偏差には、*numeric-expression* が NULL 値のローは含まれません。グループにローが含まれていない場合は、NULL を返します。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義：OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「集合関数」139 ページ
- 「WINDOW 句」989 ページ

標準と互換性

- **SQL/2008** VAR_POP 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。Window 関数として使用する場合、VAR_POP はオプションの SQL 基本機能 T611、「Elementary OLAP operations」の一部です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、VAR_POP 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の文は、異なる期間における注文ごとの項目数で平均と平方偏差をリストします。

```
SELECT YEAR( ShipDate ) AS Year,
   QUARTER( ShipDate ) AS Quarter,
   AVG( Quantity ) AS Average,
   VAR_POP( quantity ) AS Variance
FROM 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...
...	...	...	...

VAR_SAMP 関数 [集合]

数値式からなるサンプルの統計上の平方偏差を DOUBLE として計算します。

構文 1

VAR_SAMP(*numeric-expression* **)**

構文 2

VAR_SAMP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec : 以下の備考部分の構文 2 の説明を参照してください。

パラメーター

- ***numeric-expression*** ローセットでサンプルベースの平方偏差を計算する対象の式。通常、式はカラム名です。

戻り値

DOUBLE

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

numeric-expression (x) の平方偏差 (s^2) は、通常の分散と想定して、次の式によって計算されます。

$s^2 = (1/(N - 1)) * SUM(x_i - mean(x))^2$

この平方偏差には、*numeric-expression* が NULL 値のローは含まれません。グループに 0 か 1 のローが含まれている場合は、NULL を返します。

構文 2 は、SELECT 文で Window 関数として使用する場合の用法を示します。この場合、*window-spec* の要素は、関数構文の中 (インライン) に指定するか、または SELECT 文の WINDOW 句と組み合わせて指定できます。WINDOW 句の *window-spec* 定義を参照してください。

SELECT 文での Window 関数の使用方法や実例については、「Window 関数」『SQL Anywhere サーバー SQL の使用法』を参照してください。

OVER 句でのウィンドウ指定の詳細については、「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「集合関数」139 ページ
- 「VARIANCE 関数 [集合]」388 ページ
- 「WINDOW 句」989 ページ

標準と互換性

- **SQL/2008** VAR_SAMP 関数は、オプションの SQL/2008 言語機能 T621、「Enhanced numeric functions」の一部です。Window 関数として使用する場合、VAR_SAMP はオプションの SQL 基本機能 T611、「Elementary OLAP operations」の一部です。VARIANCE 構文はベンダー拡張です。

  カラム参照ではない式に対して DISTINCT を指定する機能は、オプションの SQL 言語機能 F561、「Full value expressions」の一部です。SQL Anywhere では、SQL/2008 言語機能 F441、「Extended set function support」もサポートされています。これにより、他のクエリブロックの式に対する外部参照など、カラム参照ではない任意の式を集合関数のオペランドで使用できます。

  SQL Anywhere では、オプションの SQL/2008 機能 F442、「Mixed column references in set function」がサポートされていません。SQL Anywhere では、VAR_SAMP 関数を含むクエリブロックからのカラム参照と外部参照の両方を、集合関数の引数に含めることはできません。例については、「AVG 関数 [集合]」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の文は、異なる期間における注文ごとの項目数で平均と平方偏差をリストします。

```
SELECT YEAR( ShipDate ) AS Year,
   QUARTER( ShipDate ) AS Quarter,
   AVG( Quantity ) AS Average,
   VAR_SAMP( quantity ) AS Variance
FROM 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...
...	...	...	...

VAREXISTS 関数 [その他]

指定した名前でユーザー定義変数が作成または定義されている場合は 1 を返します。該当する変数が作成されていない場合は、0 を返します。

構文

VAREXISTS(*variable-name-string* **)**

パラメーター

- ***variable-name-string*** 文字列としてテストされる変数名。

戻り値

INT

参照

- 「CREATE VARIABLE 文」679 ページ
- 「DECLARE 文」693 ページ
- 「IF 文」791 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

変数が作成または宣言されていない場合、次の IF 文は、start_time という名前の変数を作成します。作成された変数は安全に使用できます。

```
IF VAREXISTS( 'start_time' ) = 0 THEN
   CREATE VARIABLE start_time TIMESTAMP;
END IF;
SET start_time = current timestamp;
```

VARIANCE 関数 [集合]

VAR_SAMP のエイリアス。

参照

- 「VAR_SAMP 関数 [集合]」386 ページ

WATCOMSQL 関数 [その他]

Transact-SQL 文を取得し、Watcom-SQL ダイアレクトで書き換えます。この関数は、既存の Adaptive Server Enterprise ストアドプロシージャーを Watcom SQL 構文に変換するときに役立ちます。

構文

WATCOMSQL(*sql-statement-string* **)**

パラメーター

- ***sql-statement-string*** 関数で Watcom SQL ダイアレクトに書き換える SQL 文。

戻り値

LONG VARCHAR

参照

- 「SQLDIALECT 関数 [その他]」352 ページ
- 「TRANSACTSQL 関数 [その他]」373 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、文字列 'SELECT Surname as last_name FROM Employees' を返します。

```
SELECT WATCOMSQL( 'SELECT last_name = Surname FROM Employees' ) FROM dummy;
```

WEEKS 関数 [日付と時刻]

WEEKS 関数は、TIMESTAMP を操作するか、2 つの TIMESTAMP 値の間の週数を返します。次の「備考」の項を参照してください。

構文 1

WEEKS(*timestamp-expression* **)**

構文 2

WEEKS(*timestamp-expression*, *timestamp-expression* **)**

構文 3

WEEKS(*timestamp-expression*, *integer-expression* **)**

パラメーター

- ***timestamp-expression***　TIMESTAMP データ型の値の日付と時刻。
- ***integer-expression***　*timestamp-expression* に加算される週数。*integer-expression* が負の場合、*timestamp-expression*. から適切な週数が引かれます。*integer-expression* を指定する場合は、*timestamp-expression* を DATE または TIMESTAMP として明示的にキャストしてください。

戻り値

構文 1 または構文 2 では INTEGER。

構文 3 では TIMESTAMP。

備考

1 つの日付を指定すると (構文 1)、WEEKS 関数は、0000-02-29 からの週数を返します。

2 つの日付を指定すると (構文 2)、WEEKS 関数は、2 つの日付の間の週数を返します。WEEKS 関数は DATEDIFF 関数に似ていますが、2 つの日付間の週数を計算する場合に使用される方法は

これと同じではなく、返される結果が異なる場合があります。WEEKS の戻り値は、2 つの日付の間の日数を 7 で割り、余りを切り捨てて求められます。ただし、DATEDIFF では週の境目の数が使用されます。このため、2 つの関数から返される値が異なる場合があります。たとえば、最初の日付が金曜日で次の日付が翌月曜日の場合、WEEKS 関数は週数として 0 を返しますが、DATEDIFF 関数は 1 を返します。どちらかが優れているという問題ではないので、WEEKS と DATEDIFF との動作の違いをふまえて使用してください。

DATEDIFF 関数の詳細については、「DATEDIFF 関数 [日付と時刻]」196 ページを参照してください。

1 つの日付と整数を指定すると (構文 3)、WEEKS 関数は、*timestamp-expression* に、指定された整数の週数を加算します。構文 3 では、*timestamp-expression* を TIME、DATE、または TIMESTAMP データ型として明示的にキャストしてください。*timestamp-expression* が TIME 値の場合、現在の日付が使用されます。構文 3 の代わりに、DATEADD 関数を使用します。

DATEADD 関数の詳細については、「DATEADD 関数 [日付と時刻]」194 ページを参照してください。

参照

データ型のキャストの詳細については、「CAST 関数 [データ型変換]」165 ページを参照してください。

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、値 8 を返します。これは、2008-09-13 10:07:12 が、2008-07-13 06:07:12 の 8 週間後であることを示します。

```
SELECT WEEKS( '2008-07-13 06:07:12', '2008-09-13 10:07:12' );
```

次の文は、値 104792 を返します。これは、指定された日付が、0000-02-29 の 104792 週間後であることを示します。

```
SELECT WEEKS( '2008-07-13 06:07:12' );
```

次の文は、TIMESTAMP 値 2008-06-16 21:05:07.0 を返します。これは、返された日時が 2008-05-12 21:05:07 の 5 週間後であることを示します。

```
SELECT WEEKS( CAST( '2008-05-12 21:05:07' AS TIMESTAMP ), 5);
```

WRITE_CLIENT_FILE 関数 [文字列]

クライアントコンピューター上にファイルを作成して書き込みます。

構文

WRITE_CLIENT_FILE(*filename*, *blob-expression* **[, 'A'])**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- ***filename*** クライアントコンピューター上のファイル名。ファイル名は、クライアントコンピューター上で、クライアントアプリケーションの現在の作業ディレクトリとの相対パスとして解決されます。

- ***blob-expression*** クライアントコンピューター上の *filename* に書き込まれるバイナリ文字列。

- **A** デフォルトでは、ファイルがすでに存在する場合は、そのファイルが上書きされます。既存のデータにデータを追加する場合は、'A' を指定します。ファイルがまだ存在しないときに 'A' を指定した場合でも、ファイルは作成されます。

戻り値

INT

備考

データベースサーバーは、データベースの文字セットからクライアントの文字セットに *filename* を変換します。クライアントコンピューター上では、*filename* がオペレーティングシステムの文字セットに変換されます。

データがバイナリ文字列であるため、データを特定の文字セットにしたり、圧縮や暗号化を行う場合は、データにこれらの処理を実行してから、WRITE_CLIENT_FILE 関数に送信する必要があります。

クライアントのソフトウェアライブラリによってファイルの読み込みが行われ、Command Sequence 通信プロトコルを使用してデータの転送が実行されます。

パーミッション

クライアントコンピューターにあるファイルに書き込む場合

- WRITECLIENTFILE 権限が必要です。

- クライアントアプリケーションには、書き込みを行うコンピューターの書き込みパーミッションが必要です。

- allow_write_client_file データベースオプションが有効になっている必要があります。

- write_client_file セキュア機能が有効になっている必要があります。

参照

- 「WRITECLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』
- 「allow_write_client_file オプション」『SQL Anywhere サーバー データベース管理』
- 「-sf dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「クライアントコンピューター上のデータへのアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「WRITECLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』
- 「UNLOAD 文」966 ページ
- 「CSCONVERT 関数 [文字列]」189 ページ
- 「DECOMPRESS 関数 [文字列]」209 ページ
- 「DECRYPT 関数 [文字列]」211 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

XMLAGG 関数 [集合]

XML 値のコレクションから XML 要素のフォレストを生成します。

構文

XMLAGG(*expression* [**ORDER BY** *order-by-expression*] **)**

パラメーター

- ***expression***　XML 値。データ型が XML でない場合、内容はエスケープされます。*order-by-expression* は、関数が返した要素を並べ替えます。

- ***order-by-expression***　この式の値に従って XML 要素を順序付けるために使用される式。

  ORDER BY 句に定数が含まれている場合、それらの定数はオプティマイザーによって解釈され、同義の ORDER BY 句に置き換えられます。たとえば、オプティマイザーは ORDER BY 'a' を ORDER BY 式として解釈します。

  クエリブロックに、有効な ORDER BY 句が指定された複数の集合関数が含まれているとき、それらの ORDER BY 句を単一の ORDER BY 句に論理的に結合できる場合は、そのクエリブロックを実行できます。たとえば、次の ORDER BY 句の場合は、

  ```
  ORDER BY expression1, 'a', expression2
  ```

  ```
  ORDER BY expression1, 'b', expression2, 'c', expression3
  ```

  次の ORDER BY 句として結合されます。

  ```
  ORDER BY expression1, expression2, expression3
  ```

戻り値

XML

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

NULL である値はどれも結果から省かれます。すべての入力が NULL の場合、またはローがない場合、結果は NULL です。整形式の XML ドキュメントを要求する場合、生成された XML のルート要素が 1 つになるようにクエリを作成します。

XMLAGG を含むクエリを実行すると、BINARY、LONG BINARY、IMAGE、VARBINARY の各カラムのデータは、自動的に base64 エンコード形式で返されます。

ORDER BY 句を指定して XMLAGG 関数を使用するクエリの例については、「XMLAGG 関数の使用」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「XMLAGG 関数の使用」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　XMLAGG は、オプションの SQL/2008 言語機能 X034 の一部です。XMLAGG 関数のオプションの ORDER BY 句は、オプションの SQL/2008 言語機能 X035 の一部です。

例

次の文は、各顧客の注文を示す XML ドキュメントを生成します。

```
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 SalesOrders so
ORDER BY so.ID;
```

XMLCONCAT 関数 [文字列]

XML 要素のフォレストを生成します。

構文

XMLCONCAT(*xml-value* [, ...]**)**

パラメーター

- ***xml-value***　連結された XML 値。

戻り値

XML

備考

XML 要素のフォレストを生成します。解析対象外の XML ドキュメントでは、フォレストは文書内の複数のルートノードを示します。NULL 値は、結果から省かれます。値がすべて NULL の場合は、NULL が返されます。XMLCONCAT 関数は、引数にプロログがあるかどうかをチェックしません。整形式の XML ドキュメントを要求する場合、1 つのルート要素が生成されるようにクエリを作成します。

データ型が XML ではない場合、要素内容は必ずエスケープされます。XMLCONCAT 関数を含むクエリを実行すると、BINARY、LONG BINARY、IMAGE、VARBINARY の各カラムのデータは、自動的に base64 エンコード形式で返されます。

参照

- 「XMLCONCAT 関数の使用」『SQL Anywhere サーバー SQL の使用法』
- 「XMLELEMENT 関数 [文字列]」394 ページ
- 「XMLFOREST 関数 [文字列]」396 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　XMLCONCAT は、オプションの SQL/2008 言語機能 X020 の一部です。

例

次のクエリは、顧客ごとに <CustomerID>、<cust_fname>、<cust_lname> の各要素を生成します。

```
SELECT XMLCONCAT( XMLELEMENT ( NAME CustomerID, ID ),
                  XMLELEMENT( NAME cust_fname, GivenName ),
                  XMLELEMENT( NAME cust_lname, Surname )
                 ) AS "Customer Information"
FROM Customers
WHERE ID < 120;
```

XMLELEMENT 関数 [文字列]

クエリ内の XML 要素を生成します。

構文

XMLELEMENT({ **NAME** *element-name-expression* } | *string-expression*
  [, **XMLATTRIBUTES (** *attribute-value-expression* [**AS** *attribute-name*],... **)**]
  [, *element-content-expression*,...]
)

パラメーター

- ***element-name-expression***　識別子。各ローに対して、識別子と同じ名前を持った XML 要素が生成されます。

- ***attribute-value-expression***　要素の属性。このオプションの引数を使用すると、生成された要素に属性値を指定できます。この引数は、属性の名前と内容を指定します。カラム名が *attribute-value-expression* の場合、属性名はデフォルトでカラム名になります。属性名は、*attribute-name 引数を指定することにより変更できます。*

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- ***element-content-expression*** 要素の内容。任意の文字列式を指定できます。*element-content-expression* 引数は、無制限に指定でき、連結もできます。たとえば、次の SELECT 文は、値 <x>abcdef</x> を返します。

```
SELECT XMLELEMENT( NAME x, 'abc', 'def' );
```

戻り値

XML

備考

NULL の要素値と NULL の属性値は、結果から省かれます。要素名と属性名の大文字と小文字は、クエリから取得されます。

データ型が XML ではない場合、要素内容は必ずエスケープされます。無効な要素名と属性名も引用符で囲まれます。次の文を例にとります。

```
SELECT XMLELEMENT('H1', f_get_page_heading() );
```

関数 f_get_page_heading が RETURNS LONG VARCHAR または RETURNS VARCHAR(1000) と定義されている場合、結果は HTML でエンコーディングされます。

```
CREATE FUNCTION f_get_page_heading() RETURNS LONG VARCHAR
  BEGIN
    RETURN ('<B>My Heading</B>');
  END;
```

上記の SELECT 文は次の値を返します。

```
<H1>&lt;B&gt;My Heading&lt;/B&gt;</H1>
```

関数が RETURNS XML と宣言されている場合、上記の SELECT 文は次の値を返します。

```
<H1><B>My Heading</B></H1>
```

引用符と XMLELEMENT 関数の詳細については、「無効な名前と SQL/XML」『SQL Anywhere サーバー SQL の使用法』を参照してください。

XMLELEMENT 関数をネストして階層を作成できます。同じレベルのドキュメント階層で異なる要素を返したい場合に、XMLFOREST 関数を使用します。

詳細については、「XMLFOREST 関数 [文字列]」396 ページを参照してください。

XMLELEMENT 関数を含むクエリを実行すると、BINARY、LONG BINARY、IMAGE、VARBINARY の各カラムのデータは、自動的に base64 エンコード形式で返されます。

参照

- 「XMLELEMENT 関数の使用」『SQL Anywhere サーバー SQL の使用法』
- 「XMLCONCAT 関数 [文字列]」393 ページ
- 「XMLFOREST 関数 [文字列]」396 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　XMLELEMENT は、オプションの SQL/2008 言語機能 X031 の一部です。NAME キーワードを省略し、第 1 引数に文字列式を使用することは、ベンダー拡張です。SQL Anywhere は、XMLELEMENT 関数でオプションの OPTION 句をサポートしていません。

例

次の例は、結果セットの各製品に対して <item_name> 要素を生成します。ここでは、製品名が要素の内容です。

```
SELECT ID, XMLELEMENT( NAME item_name, p.Name )
FROM Products p
WHERE ID > 400;
```

次の例は、<A HREF="http://www.ianywhere.com/" TARGET="_top">iAnywhere web site</A> を返します。

```
SELECT XMLELEMENT(
  'A',
  XMLATTRIBUTES( 'http://www.ianywhere.com/'
    AS "HREF", '_top' AS "TARGET"),
  'iAnywhere web site'
);
```

次の例は、<table><tbody><tr align="center" valign="top"><td>Cell 1 info</td><td>Cell 2 info</td></tr></tbody></table> を返します。

```
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' )
         )
       )
     );
```

次の例は、'<x>abcdef</x>','<custom_element>abcdef</custom_element>' を返します。

```
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' );
```

XMLFOREST 関数 [文字列]

XML 要素のフォレストを生成します。

構文

XMLFOREST(*element-content-expression* [**AS** *element-name*],... **)**

パラメーター

- ***element-content-expression***　文字列。指定された各 *element-content-expression* 引数に対して、要素が生成されます。*element-content-expression* の値が要素の内容になります。たとえ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ば、この引数に従業員テーブルの EmployeeID カラムを指定すると、テーブルの各値に対して EmployeeID 値を含む <EmployeeID> 要素が生成されます。

要素に *element-content-expression* 以外の名前を割り当てる場合は、*element-name* 引数を指定します。それ以外の場合、要素名はデフォルトで *element-content-expression* 名になります。

戻り値

XML

備考

XML 要素のフォレストを生成します。解析対象外の XML ドキュメントでは、フォレストはドキュメント内の複数のルートノードを示します。XMLFOREST 関数の引数がすべて NULL の場合は、NULL 値が返されます。一部の値が NULL の場合、NULL 値は結果から省かれます。データ型が XML ではない場合、要素内容は必ず引用符で囲まれます。XMLFOREST 関数を使用して属性を指定することはできません。生成された要素に属性を指定する場合は、XMLELEMENT 関数を使用します。

XMLELEMENT 関数の詳細については、「XMLELEMENT 関数 [文字列]」394 ページを参照してください。

データ型が XML でない場合、要素名はエスケープされます。

整形式の XML ドキュメントを要求する場合、1 つのルート要素が生成されるようにクエリを作成します。

XMLFOREST を含むクエリを実行すると、BINARY、LONG BINARY、IMAGE、VARBINARY の各カラムのデータは、自動的に base64 エンコード形式で返されます。

参照

- 「XMLFOREST 関数の使用」『SQL Anywhere サーバー SQL の使用法』
- 「XMLELEMENT 関数 [文字列]」394 ページ
- 「XMLCONCAT 関数 [文字列]」393 ページ
- 「文字列関数」148 ページ

標準と互換性

- **SQL/2008**　XMLFOREST は、オプションの SQL/2008 言語機能 X032 の一部です。SQL Anywhere は、XMLFOREST 関数でオプションの XMLNAMESPACES 句または OPTION 句をサポートしていません。

例

次の文は、各従業員の姓と名前に対して XML 要素を生成します。

```
SELECT EmployeeID,
    XMLFOREST( GivenName, Surname )
    AS "Employee Name"
FROM Employees;
```

XMLGEN 関数 [文字列]

XQuery コンストラクターに基づいて XML 値を生成します。

構文

XMLGEN(*xquery-constructor*, *content-expression* [**AS** *variable-name*],... **)**

パラメーター

- ***xquery-constructor*** XQuery コンストラクター。XQuery コンストラクターは、XQuery 言語で定義された項目です。XQuery 式に基づいて XML 要素を構成するための構文を提供します。*xquery-constructor* 引数は、1 つ以上の変数参照を持つ整形式の XML ドキュメントにしてください。変数参照は中かっこで囲まれます。プレフィクス $ が必要で、前後に空白スペースは不要です。次に例を示します。

```
SELECT XMLGEN( '<a>{$x}</a>', 1 AS x );
```

- ***content-expression*** 変数。*content-expression* 引数は、複数指定できます。オプションの *variable-name* 引数は、変数の名前を付ける際に使用されます。次に例を示します。

```
SELECT XMLGEN( '<emp EmployeeID="{$EmployeeID}"><StartDate>{$x}</StartDate></emp>',
               EmployeeID, StartDate
               AS x )
FROM Employees;
```

戻り値

XML

備考

XQuery 仕様で定義されている計算コンストラクターは、XMLGEN 関数でサポートされません。

XMLGEN 関数を含むクエリを実行すると、BINARY、LONG BINARY、IMAGE、VARBINARY の各カラムのデータは、自動的に base64 エンコード形式で返されます。

データ型が XML ではない場合、要素内容は必ずエスケープされます。無効な XML 要素名と属性名もエスケープされます。

エスケープと XMLGEN 関数については、「無効な名前と SQL/XML」『SQL Anywhere サーバー SQL の使用法』を参照してください。

参照

- 「XMLGEN 関数の使用」『SQL Anywhere サーバー SQL の使用法』
- 「文字列関数」 148 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。XMLGEN 関数の機能は、SQL/2008 XMLDOCUMENT 関数に似ています。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例は、各従業員の <emp> 要素、<Surname> 要素、<GivenName> 要素、<StartDate> 要素を生成します。

```
SELECT XMLGEN( '<emp EmployeeID="{$EmployeeID}">
                <Surname>="{$Surname}"</Surname>
                <GivenName>="{$GivenName}"</GivenName>
                <StartDate>="{$StartDate}"</StartDate>
              </emp>',
              EmployeeID,
              Surname,
              GivenName,
              StartDate
             ) AS employee_list
FROM Employees;
```

YEAR 関数 [日付と時刻]

TIMESTAMP 引数の年部分を返します。

構文

YEAR(*timestamp-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP 値。

戻り値

SMALLINT

備考

指定された TIMESTAMP 値の年部分が SMALLINT として返されます。

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、値 2001 を返します。

```
SELECT YEAR( '2001-09-12' );
```

YEARS 関数 [日付と時刻]

YEARS 関数は、TIMESTAMP を操作するか、2 つの TIMESTAMP 値の間の年数を返します。次の「備考」の項を参照してください。

構文 1

YEARS(*timestamp-expression* **)**

構文 2

YEARS(*timestamp-expression*, *timestamp-expression* **)**

構文 2

YEARS(*timestamp-expression*, *integer-expression* **)**

パラメーター

- ***timestamp-expression*** TIMESTAMP データ型の値の日付と時刻。

- ***integer-expression*** *timestamp-expression* に (SMALLINT 値として) 追加される年数。*integer-expression* が負の場合、*timestamp-expression.* から適切な年数が引かれます。*integer-expression* を指定する場合は、*timestamp-expression* を DATE、TIME、または TIMESTAMP 値として明示的にキャストしてください。*timestamp-expression* が TIME 値の場合、現在の年が使用されます。

  データ型のキャストの詳細については、「CAST 関数 [データ型変換]」165 ページを参照してください。

戻り値

構文 1 または構文 2 では SMALLINT。

構文 3 では TIMESTAMP。

備考

YEARS の値は、2 つの日付の間に年の最初の日がいくつあるかをカウントして求められます。

参照

- 「DATEDIFF 関数 [日付と時刻]」196 ページ
- 「DATEADD 関数 [日付と時刻]」194 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文はどちらも -4 を返します。

```
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' );
```

次の文は、1998 を返します。

```
SELECT YEARS( '1998-07-13 06:07:12' )
SELECT DATEPART( year, '1998-07-13 06:07:12' );
```

次の文は、指定した日付の 300 年後を返します。

```
SELECT YEARS( CAST( '1998-07-13 06:07:12' AS TIMESTAMP ), 300 )
SELECT DATEADD( year, 300, '1998-07-13 06:07:12' );
```

YMD 関数 [日付と時刻]

指定した年、月、日に相当する日付値を返します。引数は -32768 ～ 32767 の SMALLINT です。

構文

YMD(*smallint-expression1*, *smallint-expression2*, *smallint-expression3* **)**

パラメーター

- ***smallint-expression1*** 年。
- ***smallint-expression2*** 月の数。月が 1 ～ 12 の範囲外である場合は、年が相応に調整されます。
- ***smallint-expression3*** 日の数。任意の整数を指定でき、それに応じて日付が調整されます。

戻り値

DATE

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、値 1998-06-12 を返します。

```
SELECT YMD( 1998, 06, 12 );
```

値が通常の範囲から外れている場合、それに応じて日付が調整されます。たとえば、次の文は DATE 値 2000-03-01 を返します。

```
SELECT YMD( 1999, 15, 1 );
```

SQL 文

この項では、SQL 文のマニュアルに使用する表記規則を説明します。

一般的な SQL 構文要素

この項では、多くの SQL 構文で使われる言語要素をリストします。

- ***column-name*** カラム名を表す識別子。
- ***condition*** TRUE、FALSE、または UNKNOWN の評価を行う式。
- ***connection-name*** アクティブな接続の名前を表す文字列。
- ***data-type*** 記憶データ型。
- ***expression*** 式。構文に含まれる式の一般的な例を挙げると、カラム名があります。
- ***filename*** ファイル名を指定した文字列。
- ***hostvar*** 先頭にコロンがあるホスト変数として宣言される C 言語変数の 1 つ。
- ***materialized-view-name*** マテリアライズドビュー名を表す識別子。
- ***number*** 任意の順序に並んだ数字。小数点以下の位があったり、負の記号を付けたりできます。また、数字の後に E と指数を付けることもできます。次に例を示します。

  ```
  42
  -4.038
  .001
  3.4e10
  1e-10
  ```

- ***owner*** データベースオブジェクトの所有者であるユーザー ID を表す識別子。
- ***query-block*** クエリブロックは、単純なクエリ式、または ORDER BY 句を使用したクエリ式です。
- ***query-expression*** クエリ式は、SELECT、UNION、INTERSECT、または EXCEPT ブロック (つまり ORDER BY、WITH、FOR、FOR XML、または OPTION 句を含まない文) で構成できます。これらのブロックを組み合わせて構成することも可能です。
- ***role-name*** 外部キーの役割名を表す識別子。概念データベースモデルで、ある視点からの関係を説明する動詞または句を指します。各関係は 2 つのロールを使用して表すことができます。"contains (A は B を含む)" や "is a member of (B は A のメンバー)" などのロールがあります。
- ***savepoint-name*** セーブポイント名を表す識別子。
- ***search-condition*** TRUE、FALSE、または UNKNOWN の評価を行う条件。

- ***special-value*** 特別値。
- ***statement-label*** ループまたは複合文のラベルを表す識別子。
- ***statement-list*** それぞれがセミコロンで終わる SQL 文のリスト。
- ***string-expression*** 文字列に解決される式。
- ***table-list*** テーブル名のリスト。相関名が含まれることもあります。
- ***table-name*** テーブル名を表す識別子。
- ***userid*** ユーザー名を表す識別子。
- ***variable-name*** 変数名を表す識別子。
- ***window-name*** ウィンドウ名を表す識別子。ウィンドウ定義に関する構文に使用されます (たとえば、WINDOW 句、RANK などの Window 関数)。

参照

- 「トランザクション内のセーブポイント」『SQL Anywhere サーバー SQL の使用法』
- 「変数」76 ページ
- 「真理値探索条件」58 ページ
- 「オブジェクトを所有することで取得したパーミッション」『SQL Anywhere サーバー データベース管理』
- 「マテリアライズドビュー」『SQL Anywhere サーバー SQL の使用法』
- 「Embedded SQL のホスト変数」『SQL Anywhere サーバー プログラミング』
- 「式」12 ページ
- 「SQL データ型」85 ページ
- 「文字列」5 ページ
- 「SQL Anywhere データベース接続」『SQL Anywhere サーバー データベース管理』
- 「制御文」『SQL Anywhere サーバー SQL の使用法』
- 「探索条件」34 ページ
- 「特別値」61 ページ
- 「FROM 句」758 ページ
- 「キージョイン」『SQL Anywhere サーバー SQL の使用法』
- 「識別子」4 ページ

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

- **キーワード** SQL キーワードはすべて次の例に示す SQL 文 ALTER TABLE のように大文字で表記します。

  ALTER TABLE [*owner*.]*table-name*

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **プレースホルダー**　適切な識別子または式で置き換えられる項目は、次の例に示す *owner* や *table-name* のように斜体で表記します。

  ALTER TABLE [*owner*.]*table-name*

- **句の順序**　オプション句の順序が SQL 文の構文において重要な場合、記述するべき順序でオプション句を次のように構文の本文に記述します。

  CREATE SYNCHRONIZATION SUBSCRIPTION [*subscription-name*]
  TO *publication-name*
  [**FOR** *ml-username*, ...]
  ...

  オプション句の順序が SQL 文の構文において重要ではない場合、次のようにオプション句をオプションのリストのように別々に記述します。

  CREATE [**OR REPLACE**] **SPATIAL REFERENCE SYSTEM**
  srs-name
  [*srs-attribute* [*srs-attribute* ...]

  srs-attribute :
  IDENTIFIED BY *srs-id*
  | **DEFINITION** { *definition-string* | **NULL** }
  ...

- **オプション部分**　文のオプション部分は角カッコで囲みます。次に例を示します。

  RELEASE SAVEPOINT [*savepoint-name*]

  この例では、角カッコで囲まれた *savepoint-name* がオプション部分です。角カッコは入力しないでください。

  また、キーワード部分を角カッコで囲む場合もあります。たとえば、次の構文は COMMIT TRAN または COMMIT TRANSACTION を使用できることを示します。

  COMMIT TRAN[**SACTION**] ...

  同様に、次の構文は COMMIT または COMMIT WORK を使用できることを示します。

  COMMIT [**WORK**]

- **繰り返し項目**　繰り返しできる項目は、次の例に示す *column-constraint* のように、後ろに適切なリストセパレーターと省略記号 (ピリオド 3 つ) を付けて表します。

  ADD *column-definition* [*column-constraint*, ...]

  この場合は、カラムの制約を指定しないこと、または 1 つ以上のカラムを指定することができます。複数の要素を指定する場合は、各要素間をカンマで区切ります。

- **オプション**　項目リストから 1 つだけ選択する場合や、何も選択しなくてもよい場合は、項目間をパイプ記号で区切り、リスト全体を角カッコで囲みます。

  [**ASC** | **DESC**]

この例では、ASC と DESC のどちらか 1 つを選択しても、選択しなくてもかまいません。角カッコは入力しないでください。

- **選択肢**　オプションの中の 1 つを必ず選択する場合は、選択肢を中カッコで囲みます。

  [QUOTES { ON | OFF }]

  この場合、ON または OFF のどちらかを必ず選択します。角カッコと中カッコは入力しないでください。

文の適応性インジケーター

一部の文には、タイトルの後ろに角カッコで囲まれたインジケーターが付き、文が使用される場所を示します。このインジケーターは以下のとおりです。

- **[ESQL]**　Embedded SQL で使用される文。
- **[Interactive SQL]**　Interactive SQL 専用の文。
- **[SP]**　ストアドプロシージャー、トリガー、またはバッチで使用される文。
- **[T-SQL]**　Adaptive Server Enterprise との互換性のために実装されている文。Transact-SQL フォーマット以外のストアドプロシージャーで使用できないことがあります。また、Transact-SQL の互換性が問題にならないかぎり、SQL/2008 標準に近い代わりの文が推奨される場合もあります。
- **[外部プロシージャー]**　外部ファンクションおよび外部プロシージャーの呼び出しで使用される文。
- **[Mobile Link]**　Mobile Link クライアント専用の文。
- **[SQL Remote]**　SQL Remote 専用の文。
- **[Web サービス]**　Web サービスクライアント専用の文。

カッコが 2 つある場合、文はどちらの環境でも使用できます。たとえば、[ESQL][SP] は Embedded SQL でもストアドプロシージャーでも使用できる文であることを意味します。

SQL 文

次の項では、サポートされるすべての SQL 文の構文情報を定義します。

ALLOCATE DESCRIPTOR 文 [ESQL]

SQL 記述子領域 (SQLDA) に使用する領域を割り付けます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

ALLOCATE DESCRIPTOR *descriptor-name*
[**WITH MAX** { *integer* | *hostvar* }]

descriptor-name : *identifier*

パラメーター

- **WITH MAX 句**　記述子領域の変数の数を指定できます。デフォルトのサイズは 1 です。さらに、fill_sqlda を呼び出して実際のデータ項目に使用する領域を割り付けてから、フェッチを行ったり、記述子領域内のデータにアクセスする文を実行します。

備考

記述子領域 (SQLDA) に使用する領域を割り付けます。C コードの中で次の宣言を行ってから、この文を使用します。

```
struct sqlda * descriptor_name
```

パーミッション

なし

関連する動作

なし

参照

- 「DEALLOCATE DESCRIPTOR 文 [ESQL]」684 ページ
- 「SQLDA (SQL descriptor area)」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　ALLOCATE DESCRIPTOR 文は、SQL/2008 標準のオプションの SQL 言語機能 B031、"Basic dynamic SQL" の一部です。

例

次のサンプルプログラムは、ALLOCATE DESCRIPTOR 文の使用例です。

```
#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=sql;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;
  }
```

ALTER DATABASE 文

データベースのアップグレード、データベースの jConnect サポートのオン／オフの切り替え、データベースの調整、トランザクションログファイル名とトランザクションログミラーファイル名の変更、またはミラーサーバー対するデータベースの所有権の取得の強制を実行します。

構文 1 - コンポーネントのアップグレードまたはオブジェクトのリストア

ALTER DATABASE UPGRADE
[**PROCEDURE ON**]
[**JCONNECT** { **ON** | **OFF** }]

構文 2 - 調整の実行

ALTER DATABASE {
 CALIBRATE [**SERVER**]
 | **CALIBRATE DBSPACE** *dbspace-name*
 | **CALIBRATE DBSPACE TEMPORARY**
 | **CALIBRATE GROUP READ**
 | **CALIBRATE PARALLEL READ**
 | **RESTORE DEFAULT CALIBRATION**
}

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 3 - トランザクションログ名とトランザクションログミラー名の変更

ALTER DATABASE *dbfile*
ALTER [**TRANSACTION**] **LOG** {
{ **ON** [*log-name*] [**MIRROR** *mirror-name*] | **OFF** }
[**KEY** *key*]

構文 4 - データベースの所有権の変更

ALTER DATABASE
{ *dbname* **FORCE START**
| **SET PARTNER FAILOVER** }

構文 5 - チェックサム設定の変更

ALTER DATABASE *dbfile*
CHECKSUM OFF

パラメーター

PROCEDURE 句 データベースに含まれるすべての dbo 所有プロシージャーと sys 所有プロシージャーを削除して再作成します。

JCONNECT 句 jConnect JDBC ドライバーからシステムカタログ情報にアクセスできるようにするには、JCONNECT ON を指定します。この句によって jConnect をサポートするシステムオブジェクトがインストールされます。jConnect システムオブジェクトを含まない場合、JCONNECT OFF を指定します。その場合でも、システム情報にアクセスしないかぎり、JDBC を使用できます。JCONNECT はデフォルトで ON です。

CALIBRATE [SERVER] 句 テンポラリ DB 領域以外のすべての DB 領域を調整します。この句は、CALIBRATE PARALLEL READ によって実行される処理も実行します。

CALIBRATE DBSPACE 句 指定した DB 領域を調整します。

CALIBRATE DBSPACE TEMPORARY 句 テンポラリ DB 領域を調整します。

CALIBRATE GROUP READ 句 テンポラリ DB 領域でグループ読み込みの調整を実行します。テンポラリ DB 領域に大きなワークテーブルを書き込み、さまざまなグループ読み込みサイズを使用してファイルの読み込み時間を設定します。テンポラリテーブルに追加する領域が接続の制限を超えた場合、またはキャッシュサイズが足りず、最大メモリサイズで調整できない場合、調整が失敗してエラーメッセージが返されます。

CALIBRATE PARALLEL READ 句 すべての DB 領域ファイルについてデバイスの並列 I/O 機能を調整します。CALIBRATE [SERVER] 句もこの調整を実行します。

RESTORE DEFAULT CALIBRATION 句 一般的なハードウェアと構成設定に基づく組み込みのデフォルト値に、ディスク転送時間 (DTT: Disk Transfer Time) をリストアします。

ALTER [TRANSACTION] LOG 句 トランザクションログファイルまたはトランザクションログミラーファイルの名前を変更します。MIRROR *mirror-name* を指定しないと、新しいトランザクションログのファイル名が設定されます。データベースがトランザクションログを現在使っていない場合、データベースは設定されたファイル名を使って起動します。すでにトランザクションログを使っている場合、データベースはトランザクションログとして新しいファイル名を使うように変更します。

MIRROR *mirror-name* を指定する、新しいトランザクションログミラーのファイル名として設定されます。データベースがトランザクションログミラーを現在使っていない場合、データベースはこの設定された名前を使って起動します。すでにトランザクションログミラーを使っている場合、データベースはトランザクションログミラーとして新しいファイル名を使うように変更します。

この句を使用して、トランザクションログまたはトランザクションログミラーをオフにすることもできます。たとえば、ALTER DATABASE ALTER LOG OFF のように指定します。

KEY 句　トランザクションログまたはトランザクションログミラーに使用する暗号化キーを指定します。強力に暗号化されたデータベースで ALTER [TRANSACTION] LOG 句を使用するときは、暗号化キーを指定する必要があります。

***dbname* FORCE START 句**　現在ミラーサーバーとして動作しているデータベースサーバーに、データベースの所有権の取得を強制します。この句は、プロシージャーまたはイベントから実行できます。また、ミラーサーバー上のユーティリティデータベースに接続しているときに実行する必要があります。

SET PARTNER FAILOVER 句　プライマリサーバーからミラーサーバーへのデータベースミラーリングのフェールオーバーを起動します。この文は、プライマリサーバー上のデータベースに接続しているときに実行する必要があります。プロシージャーまたはイベントから実行できます。この文を実行すると、データベースに対する既存の接続は、文を実行した接続も含めて、閉じられます。この文がプロシージャーまたはイベントに含まれている場合、後続する他の文は実行されない可能性があります。

CHECKSUM 句　データベースのグローバルチェックサムを無効にします。デフォルトでは、新しいデータベースでグローバルチェックサムが有効になりますが、バージョン 11 以前のデータベースではグローバルチェックサムは有効になりません。

リムーバブルドライブなどのストレージデバイス上で実行されているデータベースや Windows Mobile 上で実行されているデータベースでは、データベースファイルの破損を速やかに検出できるように、この句の設定に関係なく、データベースサーバーによって常に書き込みチェックサムが有効になります。また、検証アクティビティの実行時に重要なページのチェックサムも計算されます。

グローバルチェックサムが有効になっていないデータベースでは、-wc オプションを使用して書き込みチェックサムを有効にすることができます。

備考

- **構文 1**　ALTER DATABASE UPGRADE 文は、データベースをアップグレードまたは更新するためのアップグレードユーティリティの代わりとして使用できます。リリースを維持するときにも適用されます。この文を実行した後は、データベースを再起動してください。一般に、マイナーバージョンでのデータベースの変更はデータベースオプションの追加やシステムテーブルとプロシージャーの細かい変更にかぎられます。ALTER DATABASE UPGRADE 文は、システムテーブルを最新バージョンにアップグレードし、新規データベースオプションを追加します。必要に応じて、すべてのシステムプロシージャーを削除し、再作成します。PROCEDURE ON 句を指定すると、システムプロシージャーを強制的に再構築できます。

  ALTER DATABASE UPGRADE 文を現在ミラーされているデータベースで実行すると、エラーメッセージが返されます。

ALTER DATABASE UPGRADE 文を使用して、設定やシステムオブジェクトを元のインストール状態にリストアすることもできます。

データベースファイルの物理的な再編成を必要とする機能は、ALTER DATABASE UPGRADE 文を実行して使用可能にすることはできません。そのような機能には、インデックスの拡張やデータの格納に関する変更が含まれています。これらの拡張機能を利用するには、データベースのアンロードと再ロードを行ってください。

警告
データベースをアップグレードする前に、データベースファイルをバックアップすることを強くおすすめします。アップグレード処理に失敗した場合、部分的にアップグレードされたデータベースには無効のマークが付き、使用できなくなります。アップグレードに失敗した後にデータベースを起動しようとすると、データベースファイルが無効であることを示すエラーが表示されます。

jConnect JDBC ドライバーを使用してシステムカタログ情報にアクセスするには、JCONNECT ON (デフォルト値) を指定します。jConnect システムオブジェクトを含まない場合、JCONNECT OFF を指定します。JCONNECT OFF を設定しても、データベースから jConnect サポートが削除されることはありません。その場合でも、システムカタログ情報にアクセスしないかぎり、JDBC を使用できます。続けて jConnect の新しいバージョンをダウンロードする場合、ALTER DATABASE UPGRADE JCONNECT ON 文を (再) 実行して、データベースのバージョンをアップグレードできます。

- **構文 2**　構文 2 を使用すると、オプティマイザーが使用する I/O コストモデルを再調整することもできます。この操作により、ディスク転送時間 (DTT: Disk Transfer Time) モデルが更新されます。DTT は、コストモデルで使用されるディスク I/O の数学的モデルです。I/O コストモデルを再調整するときは、データベースサーバーを他の用途に使用できません。また、コンピューター上の他のすべてのアクティビティがアイドル状態になっている必要があります。データベースサーバーの再調整は高負荷のオペレーションであり、完了までに長時間かかることがあります。通常はデフォルトのままにすることをおすすめします。

  CALIBRATE PARALLEL READ 句を使用する場合、10000 ページ未満の DB 領域ファイルでは並列の調整は実行されません。調整操作中にデータベースサーバーが自動的にすべてのアクティビティを中断した場合でも、同じコンピューターに大量のリソースを消費するプロセスがなければ、並列の調整は実行されます。調整後、IOParallelism 拡張データベースプロパティを使用して、DB 領域ファイルに使用できる並列 I/O 操作の最大推定数を取得できます。

  同じようなハードウェアインストールが大量にある場合に、繰り返し行われる時間のかかる再調整作業を省くには、sa_unload_cost_model システムプロシージャーを使用してアンロードしてから、sa_load_cost_model システムプロシージャーを使用して別のデータベースに適用 (ロード) して調整を再使用できます。

- **構文 3**　ALTER DATABASE 文を使用して、データベースファイルに関連付けられているトランザクションログとトランザクションログミラーの名前を変更できます。実行中のデータベースは変更できません。これらの変更は、トランザクションログ (dblog) ユーティリティで行う変更と同じです。この文は、-gu オプションの設定に応じて、ユーティリティデータベースまたは別のデータベースと接続しているときに実行できます。暗号化されたデータベースのトランザクションログまたはトランザクションログミラーを変更する場合は、キーを指定してください。データベースが監査を実行中にトランザクションログの使用を停止すること

はできません。監査をオフにすると、トランザクションログの使用を停止できます。この構文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

BACKUP DATABASE 文を使用して、実行中のデータベースのトランザクションログの名前を変更できます。次に例を示します。

```
BACKUP DATABASE DIRECTORY 'directory-name'
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

- **構文 4**　ALTER DATABASE...FORCE START は、プライマリサーバーではなくミラーサーバーから実行する必要があります。ミラーリングされていないデータベースや現在アクティブでこのサーバーに所有されているデータベースに対して ALTER DATABASE...FORCE START 文を実行しようとすると、エラーになります。また、プライマリサーバーがまだミラーサーバーに接続されている場合も、エラーになります。

- **構文 5**　この句は、データベースのチェックサムを無効にする場合にのみ使用できます。

パーミッション

構文 1 と 2 の場合、DBA 権限が必要です。また、このデータベースに他の接続がないことが必要です。ALTER DATABASE UPGRADE 文は、Windows Mobile ではサポートされません。

構文 3 の場合、トランザクションログが保存されているディレクトリへのファイルパーミッションが必要です。また、データベースが実行中ではない必要があります。

構文 4 の場合、実行するユーザーには、-gk サーバーオプションで指定されているパーミッションが必要です。

構文 5 の場合、DBA 権限が必要です。

関連する動作

オートコミット。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CREATE DATABASE 文」521 ページ
- 「CREATE STATISTICS 文」643 ページ
- 「BACKUP 文」489 ページ
- 「アップグレードユーティリティ (dbupgrad)」『SQL Anywhere サーバー データベース管理』
- 「トランザクションログユーティリティ (dblog)」『SQL Anywhere サーバー データベース管理』
- 「DB_EXTENDED_PROPERTY 関数 [システム]」203 ページ
- 「-gk dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「-gu dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「-wc dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「-wc dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「sa_validate システムプロシージャー」1210 ページ
- 「sa_unload_cost_model システムプロシージャー」1206 ページ
- 「sa_load_cost_model システムプロシージャー」1109 ページ
- 「データベースサーバーの強制プライマリサーバー化」『SQL Anywhere サーバー データベース管理』
- 「プライマリサーバーのフェールオーバー」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリング」『SQL Anywhere サーバー データベース管理』
- 「データベースの再構築」『SQL Anywhere サーバー SQL の使用法』
- 「データベースミラーリングシステムでのデータベースのアップグレード」『SQL Anywhere 12 変更点とアップグレード』
- 「DB_EXTENDED_PROPERTY 関数 [システム]」203 ページ
- 「jConnect システムオブジェクトのデータベースへのインストール」『SQL Anywhere サーバー プログラミング』
- 「チェックサムを使用した破損の検出」『SQL Anywhere サーバー データベース管理』
- 「Windows Mobile の jConnect」『SQL Anywhere サーバー データベース管理』
- 「検証ユーティリティ (dbvalid)」『SQL Anywhere サーバー データベース管理』
- 「VALIDATE 文」983 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　ALTER DATABASE 文は、Adaptive Server Enterprise でサポートされています。ただし、Adaptive Server Enterprise でサポートされる文の句は、SQL Anywhere でサポートされる句とは切り離されています。

例

次の例では、jConnect サポートを無効にします。

```
ALTER DATABASE UPGRADE JCONNECT OFF;
```

次の例は、*demo.db* に関連するトランザクションログファイル名を *mynewdemo.log* に設定します。

```
ALTER DATABASE 'demo.db'
  ALTER LOG ON 'mynewdemo.log';
```

ALTER DBSPACE 文

DB 領域またはトランザクションログ用に領域を事前に割り付けたり、DB 領域ファイルの名前変更時や移動時にカタログを更新したりします。

構文

ALTER DBSPACE { *dbspace-name* | **TRANSLOG** | **TEMPORARY** }
{ **ADD** *number* [*add-unit*]
| **RENAME** *filename* }

add-unit :
PAGES
| **KB**
| **MB**
| **GB**
| **TB**

パラメーター

TRANSLOG 句　特別な DB 領域名 TRANSLOG を指定して、トランザクションログにディスク領域を事前に割り付けます。 事前に割り付けておくと、トランザクションログが急速に大きくなることが予測される場合に、パフォーマンスを改善できます。 たとえば、ビットマップのような多量のバイナリラージオブジェクト (BLOB) を処理する場合、この機能を使用できます。

構文 **ALTER DBSPACE** *dbspace-name* **TRANSLOG RENAME** *filename* はサポートされていません。

TEMPORARY 句　特別な DB 領域名 TEMPORARY を指定して、テンポラリ DB 領域にスペースを追加します。テンポラリ DB 領域にスペースが追加されるとすぐに、追加のスペースは対応するテンポラリファイル内で実体化されます。 データベースのテンポラリ DB 領域に領域を事前に割り付けると、大きなワークテーブルを使用する複雑なクエリを実行する場合、パフォーマンスが向上します。

ADD 句　ALTER DBSPACE 文に ADD 句を指定して、DB 領域にディスク領域を事前に割り付けます。ページ、キロバイト (KB)、メガバイト (MB)、ギガバイト (GB)、またはテラバイト (TB) 単位でサイズを指定して、対応するデータベースファイルを拡張します。 単位を指定しない場合、デフォルトは PAGES です。 データベースのページサイズはデータベースの作成時に決定されます。

領域が事前に割り付けられていない場合、データベースファイルは、領域が必要になったとき、ページサイズが 2 KB、4 KB、8 KB の場合は一度に約 256 KB 拡張され、その他のページサイズの場合は約 32 ページ拡張されます。 領域を事前に割り付けると、多量のデータをロードする場合のパフォーマンスを改善でき、ファイルシステム内でデータベースファイルの断片化を防ぐことができます。

この句を使用して、事前定義の DB 領域 (system、temporary、temp、translog、translogmirror) のいずれかに領域を追加できます。

RENAME 句　メインファイル以外のデータベースファイルを別のファイル名に変更したり、別のディレクトリまたはデバイスに移動したりする場合は、RENAME 句を指定した ALTER

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DBSPACE 文を使用すると、SQL Anywhere にデータベース起動時に確実に新しいファイルを検索させることができます。 *filename* パラメーターには、文字列リテラルまたは変数を指定できます。

「クラウドに関する注意点」クラウドのテナントデータベースについては、DB 領域の位置を指定する場合には、ファイル名だけを指定できます。ディレクトリパスを指定することはできません。

名前の変更は、次のように有効になります。

○文を実行する前に DB 領域がすでに開いている場合 (つまり、実際のファイルの名前はまだ変更していない場合)、継続してアクセスすることはできますが、カタログに格納されている名前は更新されます。 データベースが停止した後、ファイルの名前を変更して、RENAME 句で指定した名前と同じ名前に変更する必要があります。そうしないと、カタログにある DB 領域の名前とファイル名が一致せず、データベースを次に起動するときに、データベースサーバーが DB 領域を開くことができなくなります。

○文を実行したときに DB 領域が開いていない場合、データベースサーバーは、カタログを更新し、その後で DB 領域を開くことを試行します。 DB 領域を開くことができたら、アクセス可能になっています。 DB 領域を開くことができない場合でも、エラーは返されません。

DB 領域が開いているかどうかを確認するには、次に示す文を実行します。結果が NULL である場合は、DB 領域は開いていません。

```
SELECT DB_EXTENDED_PROPERTY('FileSize','dbspace-name');
```

メイン DB 領域の system に RENAME 句を指定した ALTER DBSPACE を使用しても効果はありません。RENAME 句はトランザクションログファイルの名前の変更には対応していません。 BACKUP DATABASE 文を使用して、実行中のデータベースのトランザクションログの名前を変更できます。 次に例を示します。

```
BACKUP DATABASE DIRECTORY 'directory-name'
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

備考

それぞれのデータベースは 1 つまたは複数のファイルの中に保持されます。DB 領域は、各データベースファイルに関連付けられた論理名を持つ追加ファイルであり、メインデータベースファイル単独では保持できないデータを格納するために使用されます。 ALTER DBSPACE は、メイン DB 領域 (ルートファイルとも呼ばれます) または追加の DB 領域を変更します。 データベースの DB 領域名は、SYSDBSPACE ビューに保持されます。メインデータベースファイルの DB 領域名は system です。

マルチファイルデータベースの起動時には、開始行または ODBC データソースの記述によって、SQL Anywhere にメインデータベースファイルの場所が示されます。 メインデータベースファイルは、システムテーブルを保持しており、SQL Anywhere は、このシステムテーブルを調べて他の DB 領域のロケーションを検索します。 次に、SQL Anywhere は、検索した各 DB 領域を開きます。 default_dbspace オプションを設定して新規テーブルを作成する DB 領域を指定できます。

パーミッション

DBA 権限が必要です。また、このデータベースに他のユーザーがいないことが必要です。

関連する動作

オートコミット。

参照

- 「CREATE DBSPACE 文」528 ページ
- 「BACKUP 文」489 ページ
- 「SYSDBSPACE システムビュー」1260 ページ
- 「default_dbspace オプション」『SQL Anywhere サーバー データベース管理』
- 「事前定義の DB 領域」『SQL Anywhere サーバー データベース管理』
- 「データベースファイルのタイプ」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、system の DB 領域サイズを 200 ページ増やします。

```
ALTER DBSPACE system
ADD 200;
```

次の例は、system の DB 領域サイズを 400 MB 増やします。

```
ALTER DBSPACE system
ADD 400 MB;
```

次の例は、system_2 の DB 領域に関連するファイル名を変更します。

```
ALTER DBSPACE system_2
RENAME 'e:¥db¥dbspace2.db';
```

ALTER DOMAIN 文

ユーザー定義のドメインまたはデータ型の名前を変更します。

構文

ALTER { **DOMAIN** | **DATATYPE** } *user-type*
RENAME *new-name*

備考

この文を実行すると、ユーザー定義のドメインまたはデータ型の名前が ISYSUSERTYPE システムテーブル内で更新されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **注意**
> このユーザー定義ドメインまたはデータ型を参照するすべてのプロシージャー、トリガー、ビュー、またはイベントを再作成してください。再作成しないと、引き続き古い名前が参照されます。

パーミッション

DBA 権限があるか、ドメインを作成したデータベースユーザーであることが必要です。

関連する動作

オートコミット。

参照

- 「SYSUSERTYPE システムビュー」1330 ページ
- 「CREATE DOMAIN 文」533 ページ
- 「ドメイン」125 ページ
- 「ドメイン」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。ALTER DOMAIN 文は、SQL/2008 標準のオプションの SQL 機能 F711 です。ただし、標準では、ALTER DOMAIN では、変更された DEFAULT または CHECK 制約句を既存のドメインに対して指定できます。SQL Anywhere では、このいずれの操作もサポートされていません。機能 F711 では、ドメイン名の変更はサポートされていません。

例

次の例は、Address ドメインの名前を MailingAddress に変更します。

```
ALTER DOMAIN Address RENAME MailingAddress;
```

ALTER EVENT 文

イベントの定義、またはイベントに関連付けて定義済みアクションを自動化するイベントハンドラーの定義を変更したり、スケジュールされたアクションの定義を変更したりします。この文を使用して、イベントハンドラーの定義を隠すこともできます。

構文 1 - イベントの変更

ALTER EVENT [*owner*.]*event-name*
[**AT** { **CONSOLIDATED** | **REMOTE** | **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*

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*

構文 2 - イベントハンドラーの定義を隠す

ALTER EVENT *event-name* **SET HIDDEN**

パラメーター

AT 句　この句は、イベントを処理するデータベースに関する指定を変更するときに使用します。

DELETE TYPE 句　この句は、イベントとイベントタイプの関連付けを解除するときに使用します。

ADD | ALTER | DELETE SCHEDULE 句　この句は、スケジュールの定義を変更するときに使用します。1 つの ALTER EVENT 文で 1 つのスケジュールしか変更できません。

WHERE 句　この句は、イベント発生のトリガー条件を変更するときに使用します。WHERE NULL オプションは条件を削除します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

START TIME 句　この句は、イベントの開始時刻とオプションで終了時刻を指定するときに使用します。*start-time* パラメーターと *end-time* パラメーターは文字列です (たとえば、'12:34:56')。変数と式は使用できません (たとえば、NOW())。

START DATE 句　この句は、イベントの開始日を指定するときに使用します。*start-date* パラメーターは文字列です。変数と式は使用できません (たとえば、TODAY())。

SET HIDDEN 句　この句は、イベントハンドラーの定義を隠すときに使用します。SET HIDDEN 句を指定すると、ISYSEVENT システムテーブルの action カラムに格納されているイベントハンドラーの定義が永続的に難読化されます。

備考

この文によって、CREATE EVENT で作成されたイベント定義を変更することができます。この文は以下の目的に使用できます。

- イベントハンドラーの定義を非表示にする
- 開発段階でトリガー条件やスケジュールを指定せずにイベントハンドラーを定義、テストし、イベントハンドラーの完成後に ALTER EVENT を使って実行条件を追加する

イベントを変更する必要がある場合は、ALTER EVENT...DISABLE 文を実行すると実行中のイベントを無効にできます。Sybase Central でイベントを無効にするには、イベントを右クリックし、**[有効化]** オプションをクリアします。イベントを無効にしても、現在のイベントハンドラーの実行は中断されません。イベントハンドラーの実行は、完了するまで続行されます。完了したイベントハンドラーは、再度有効に設定されるまで再開されません。定義は、変更して再度有効にすることができます。実行中のイベントを確認するには、次の文を実行します。

```
SELECT *
FROM dbo.sa_conn_info()
WHERE CONNECTION_PROPERTY( 'EventName',Number ) = 'event-name';
```

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「システムイベント」『SQL Anywhere サーバー データベース管理』
- 「SYSEVENT システムビュー」1263 ページ
- 「BEGIN 文」497 ページ
- 「CREATE EVENT 文」540 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

ALTER EXTERNAL ENVIRONMENT 文

Java、PHP、Perl などの外部環境のロケーションを指定します。

構文

ALTER EXTERNAL ENVIRONMENT *environment-name*
LOCATION *location-string*

environment-name :
JAVA
| PERL
| PHP
| CLR
| C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64
| DBMLSYNC

パラメーター

environment-name *environment-name* は、変更する外部環境を指定するときに使用します。

LOCATION 句 LOCATION 句は、外部環境の実行プログラム／バイナリを検出できるデータベースサーバーコンピューター上のロケーションを指定するときに使用します。この句には、実行プログラム／バイナリの名前を指定します。このパスは、完全修飾パスまたは相対パスのいずれかにできます。相対パスの場合、実行プログラム／バイナリは、サーバーが検出できるロケーションに存在する必要があります。

備考

dbmlsync ユーティリティのロケーションは、この文を使用して設定できます。同期中にデータベースサーバーの PATH 環境変数を使用して、dbmlsync の実行可能イメージを見つけられない場合は、この文を使用して、実行可能ファイルのロケーションを記録します。

パーミッション

DBA 権限。

関連する動作

なし

参照

- 「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』
- 「START EXTERNAL ENVIRONMENT 文」939 ページ
- 「STOP EXTERNAL ENVIRONMENT 文」947 ページ
- 「INSTALL EXTERNAL OBJECT 文」807 ページ
- 「REMOVE EXTERNAL OBJECT 文」875 ページ
- 「SYSEXTERNENV システムビュー」1265 ページ
- 「SYSEXTERNENVOBJECT システムビュー」1266 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、Perl を外部環境として使用するときに使用する Perl 実行ファイルを指定します。

```
ALTER EXTERNAL ENVIRONMENT PERL
LOCATION 'c:¥¥Perl64¥¥bin¥¥perl.exe';
```

ALTER FUNCTION 文

関数を変更します。

構文 1 - 関数定義の変更

ALTER FUNCTION [*owner*.]*function-name function-definition*

function-definition : *CREATE FUNCTION 構文*

構文 2 - 関数定義の難読化

ALTER FUNCTION [*owner*.]*function-name*
SET HIDDEN

構文 3 - 関数の再コンパイル

ALTER FUNCTION [*owner*.]*function-name*
RECOMPILE

備考

新しい関数全体を ALTER FUNCTION 文にインクルードします。

- **構文 1**　ALTER FUNCTION 文の構文は、最初の 1 語を除き、CREATE FUNCTION 文の構文とまったく同じです。

  ALTER FUNCTION では、関数の既存のパーミッションは変更されません。ただし、DROP FUNCTION に続けて CREATE FUNCTION を実行した場合は、実行パーミッションが再割り当てされます。

- **構文 2**　SET HIDDEN を使用して、関連する関数定義を難読化し、解読できないようにします。この関数はアンロードして、他のデータベースに再ロードできます。

  SET HIDDEN を使用すると、デバッガーを使用したデバッグでも、プロシージャープロファイリングによっても、関数定義は表示されません。

  > **注意**
  > **この設定は、元に戻せません。**元の関数定義をデータベースの外部に保持することを強くおすすめします。

- **構文 3**　RECOMPILE 構文を使用して、ユーザー定義 SQL 関数を再コンパイルします。関数を再コンパイルすると、カタログに格納された定義が再解析され、構文が検証されます。保

持されている関数のソースは、再コンパイルしても変わりません。関数を再コンパイルすると、SET HIDDEN 句で難読化された定義は、難読化された状態が維持され、解読できません。

パーミッション

関数の所有者であるか、DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [外部呼び出し]」550 ページ
- 「CREATE FUNCTION 文 [Web サービス]」555 ページ
- 「ALTER PROCEDURE 文」433 ページ
- 「DROP FUNCTION 文」714 ページ
- 「プロシージャー、ファンクション、トリガー、イベント、ビューの内容を隠す」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** ベンダー拡張。ALTER FUNCTION は、SQL/2008 標準のオプションの SQL 言語機能 F381 です。ただし、SQL 標準では、PSM 関数定義を再定義するために ALTER FUNCTION を使用することはできません。SQL/2008 では、SET HIDDEN または RECOMPILE はサポートされていません。

例

この例では、MyFunction が作成され、変更されます。SET HIDDEN 句は、関数を難読化し、解読できないようにします。

```
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;
```

ALTER INDEX 文

インデックス、プライマリキー、または外部キーの名前を変更したり、インデックスのクラスター化された内容を変更したりします。

構文

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**

パラメーター

rename-clause インデックス、プライマリキー、または外部キーの新しい名前を指定します。

外部キーまたはプライマリキーの基本となるインデックスの名前を変更しても、インデックスの対応する RI 制約の名前は変更されません。ただし、外部キーの役割名 (該当する場合) はインデックス名と同じであり、変更されます。必要に応じて ALTER TABLE 文を使用して、RI 制約名を変更します。

cluster-clause インデックスを CLUSTERED と NONCLUSTERED のどちらに変更するかを指定します。特定のテーブル上で 1 つのインデックスのみ、クラスター化できます。

REBUILD 句 この句は、インデックスを削除して再作成するのではなく、インデックスを再構築するときに使用します。

備考

ALTER INDEX 文は、次の 2 つのタスクを実行します。

- インデックス、プライマリキー、または外部キーの名前を変更するときに使用します。
- インデックスタイプをノンクラスタードからクラスタードに、またはその逆に変更するときに使用します。

  ALTER INDEX 文は、インデックスのクラスタ指定の変更には使用できますが、データの再編成はしません。また、特定のテーブルまたはマテリアライズドビュー上で 1 つのインデックスのみ、クラスター化できます。

ローカルテンポラリテーブル上では ALTER INDEX を使用してインデックスを変更できません。この文を使用してインデックスを削除しようとすると、"インデックスが見つかりません。" エラーになります。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、この文は実行できません。

パーミッション

テーブル所有者であるか、テーブルまたはマテリアライズドビューに対する REFERENCES パーミッションまたは DBA 権限が必要です。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。現在接続しているすべてのカーソルを閉じます。ALTER INDEX REBUILD が指定されている場合は、チェックポイントが実行されます。

参照

- 「CREATE INDEX 文」567 ページ
- 「ALTER TABLE 文」462 ページ
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、Products テーブルのインデックス IX_product_name の名前を ixProductName に変更します。

```
ALTER INDEX IX_product_name ON Products
RENAME TO ixProductName;
```

次の文は、IX_product_name をクラスタードインデックスに変更します。

```
ALTER INDEX IX_product_name ON Products
CLUSTERED;
```

ALTER LOGIN POLICY 文

既存のログインポリシーを変更します。

構文

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* }

パラメーター

policy-name　ログインポリシーの名前。ルートを指定してルートログインポリシーを修正します。

policy-option-name　ポリシーオプションの名前。

policy-option-value　ログインポリシーオプションに割り当てられている値。UNLIMITED を指定すると、制限は使用されません。DEFAULT を指定すると、デフォルトの制限が使用されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

ログインポリシーが変更されるとすぐに、すべてのユーザーに変更が適用されます。

ポリシーオプションを指定しない場合は、ルートログインポリシーからログインポリシーの値が取得されます。

すべての新しいデータベースには、ルートログインポリシーが含まれています。ルートログインポリシーの値を変更することはできますが、ポリシーは削除できません。次の表は、ルートログインポリシーのデフォルトオプションの概要を示します。

policy-option-name	説明	デフォルト値	適用対象
password_life_time	パスワードの変更が必要となるまでの最大日数。	Unlimited	すべてのユーザー (DBA 権限を持つユーザーを含む)。
password_grace_time	パスワードの有効期限が切れるまでの日数。この期間中は、ログインは許可されますが、デフォルトの post_login プロシージャーから警告が出ます。	0	すべてのユーザー (DBA 権限を持つユーザーを含む)。
password_expiry_on_next_login	このオプションの値が ON の場合、次回のログイン後にユーザーのパスワードは期限切れになります。	OFF	すべてのユーザー (DBA 権限を持つユーザーを含む)。
locked	このオプションの値が ON の場合、ユーザーは新しい接続を確立できません。DBA 権限を持つユーザーをロックすることはできません。sa_get_user_status システムプロシージャーの reason_locked カラムは、データベースサーバーで生成されたユーザーのロック理由を示す文字列を返します。	OFF	DBA 権限を持たないユーザー。

policy-option-name	説明	デフォルト値	適用対象
max_connections	ユーザーに許容される同時接続の最大数。	Unlimited	DBA 権限を持たないユーザー。
max_failed_login_attempts	前回のログイン成功以降、ユーザーがロックされるまでのログイン失敗の最大回数。	Unlimited	すべてのユーザー (DBA 権限を持つユーザーを含む)。 最後のログイン失敗から 1 分が経過すると、DBA ユーザーアカウントのロックは解除されます。
max_days_since_login	同一ユーザーによる 2 回の連続するログインの間で許容される最大経過日数。	Unlimited	DBA 権限を持たないユーザー。
max_non_dba_connections	DBA 権限を持たないユーザーが確立できる同時接続の最大数。このオプションは、ルートログインポリシーでのみサポートされます。	Unlimited	DBA 権限を持たないユーザー。ルートログインポリシーのみ。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「ログインポリシーの変更」『SQL Anywhere サーバー データベース管理』
- 「ALTER USER 文」482 ページ
- 「COMMENT 文」512 ページ
- 「CREATE LOGIN POLICY 文」573 ページ
- 「CREATE USER 文」677 ページ
- 「DROP LOGIN POLICY 文」716 ページ
- 「DROP USER 文」735 ページ
- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例は、Test1 ログインポリシーを変更します。この例は、locked オプションと max_connections オプションを変更します。locked の値は、このポリシーを割り当てられたユーザーが新しい接続を確立できないことを示し、max_connections の値は、許容される同時接続数を制限します。

```
ALTER LOGIN POLICY Test1
locked=ON
max_connections=5;
```

ALTER MATERIALIZED VIEW 文

マテリアライズドビューを変更します。

構文

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 : *整数*

パラメーター

SET HIDDEN 句　SET HIDDEN 句は、マテリアライズドビューの定義を難読化するときに使用します。**この設定は、元に戻せません**。

ENABLE 句　ENABLE 句は、無効にされたマテリアライズドビューを有効にして、データベースサーバーで使用できるようにするときに使用します。この句は、すでに有効になっているビューには影響ありません。この句を使用した後、ビューをリフレッシュして初期化し、ビューが無効にされたときに削除されたすべてのテキストインデックスを再作成してください。

DISABLE 句　DISABLE 句は、データベースサーバーからビューを使用できないようにします。マテリアライズドビューを無効にすると、データベースサーバーはビューのデータとインデックスを削除します。

{ ENABLE | DISABLE } USE IN OPTIMIZATION 句　この句は、オプティマイザーでマテリアライズドビューを使用できるようにするかどうかを指定するときに使用します。DISABLE USE IN OPTIMIZATION を指定する場合、マテリアライズドビューを使用するのは、明示的にそのビューを参照するクエリを実行する場合のみです。デフォルトは ENABLE USE IN OPTIMIZATION です。

ADD PCTFREE 句　各ページに確保する空き領域の割合を指定します。空き領域は、データが更新されたときにローのサイズが増えた場合に使用されます。ページに空き領域がない場合は、ページのローのサイズが増えるたびに、ローを複数のページに分割することが必要になり、ローの断片化が発生します。また、パフォーマンス低下の可能性があります。

percent-free-space の値は、0 ～ 100 までの整数です。0 を指定すると、各ページに空き領域を残さず、各ページを完全にパックします。高い値に設定すると、各ローは単独でページに挿入されます。PCTFREE が設定されない場合、または削除された場合、データベースのページサイズに応じたデフォルトの PCTFREE 設定が適用されます (ページサイズが 4 KB の場合は 200 バイト、2 KB の場合は 100 バイト)。

DROP PCTFREE 句　現在のマテリアライズドビューで有効な PCTFREE 設定を削除し、データベースのページサイズに応じたデフォルトの PCTFREE を適用します。

[NOT] ENCRYPTED 句　マテリアライズドビューデータを暗号化するかどうかを指定します。デフォルトで、マテリアライズドビューデータは作成時に暗号化されません。マテリアライズドビューを暗号化するには、ENCRYPTED を指定します。マテリアライズドビューを復号化するには、NOT ENCRYPTED を指定します。

REFRESH 句　REFRESH 句は、マテリアライズドビューの再表示タイプを変更するときに使用します。

○ **IMMEDIATE REFRESH**　IMMEDIATE REFRESH 句は、手動ビューを即時ビューに変更するときに使用します。再表示タイプを IMMEDIATE REFRESH に変更するには、手動ビューが有効であり、初期化されていない必要があります。ビューが初期化された状態である場合は、TRUNCATE 文を実行して初期化されていない状態に変更してから、ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH を実行します。

○ **MANUAL REFRESH**　MANUAL REFRESH 句は、即時ビューを手動ビューに変更するときに使用します。

備考

別のユーザーが所有するマテリアライズドビューを変更する場合は、所有者を含めて名前を修飾する必要があります (たとえば、GROUPO.EmployeeConfidential)。名前を修飾しなかった場合、データベースサーバーは、ユーザー本人が所有する同名のマテリアライズドビューを検索して変更します。見つからない場合は、エラーが返されます。

マテリアライズドビューを無効にすると (DISABLE 句)、データベースサーバーがクエリに応答するときにマテリアライズドビューを使用できなくなります。また、データとインデックスが削除され、再表示タイプが手動に変わります。通常の従属ビューもすべて無効にされます。

DISABLE 句によって従属ビューも無効になるため、無効化されるビューだけでなく、すべての従属ビューに対する排他アクセスが必要です。

マテリアライズドビューを暗号化するには (ENCRYPTED 句)、データベースでテーブルの暗号化をあらかじめ有効にしておく必要があります。その後、データベースの作成時に指定された暗号化キーとアルゴリズムを使用してマテリアライズドビューが暗号化されます。

パーミッション

ALTER MATERIALIZED VIEW 文を実行するには、ビューを所有しているか、DBA 権限を持っている必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DBA 権限がない場合に、マテリアライズドビューを即時ビューに変更するときは (ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH)、ビューとそのすべての参照先テーブルを所有している必要があります。

ユーザーが自分のデータを変更するためにマテリアライズドビューに対して実行できる操作は、リフレッシュ、トランケート、無効化のみです。ただし、即時ビューは、データベースサーバーによって自動的に更新されます。つまり、一度即時ビューを有効にして初期化すると、データベースサーバーはさらにパーミッションをチェックすることなく、ビューを自動的に管理します。

関連する動作

オートコミット。

参照

- 「CREATE MATERIALIZED VIEW 文」576 ページ
- 「REFRESH MATERIALIZED VIEW 文」867 ページ
- 「DROP MATERIALIZED VIEW 文」717 ページ
- 「TRUNCATE 文」961 ページ
- 「sa_refresh_materialized_views システムプロシージャー」1158 ページ
- 「マテリアライズドビュー」『SQL Anywhere サーバー SQL の使用法』
- 「ビューの依存性」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューを手動から即時に変更する際の制限」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューの有効化または無効化」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューの暗号化または復号化」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューを隠す」『SQL Anywhere サーバー SQL の使用法』
- 「再表示タイプの手動または即時への設定」『SQL Anywhere サーバー SQL の使用法』
- 「高度：マテリアライズドビューのステータスとプロパティ」『SQL Anywhere サーバー SQL の使用法』
- 「オプティマイザーによるマテリアライズドビューの使用の有効化または無効化」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、EmployeeConfid88 というマテリアライズドビューを作成し、その最適化での使用を無効にします。

```
CREATE MATERIALIZED VIEW EmployeeConfid88 AS
  SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary, ManagerID,
    Departments.DepartmentName, Departments.DepartmentHeadID
  FROM Employees, Departments
  WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid88;
ALTER MATERIALIZED VIEW GROUPO.EmployeeConfid88 DISABLE USE IN OPTIMIZATION;
```

警告
この例を実行し終わったら、作成したマテリアライズドビューを削除してください。このようにしないと、他の例を試すとき、基本となるテーブル Employees および Departments に対するスキーマ変更を実行できなくなります。有効化されている従属マテリアライズドビューを持つテーブルのスキーマは変更できません。

ALTER MIRROR SERVER 文

注意
読み込み専用のスケールアウトとデータベースミラーリングには、それぞれ別のライセンスが必要です。 「別途ライセンスが必要なコンポーネント」『SQL Anywhere 12 紹介』を参照してください。

ミラーサーバーの属性を変更します。

構文

ALTER MIRROR SERVER *mirror-server-name*
AS { **PRIMARY** | **MIRROR** | **ARBITER** | **PARTNER** | **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

パラメーター

- **AS 句** 次のいずれかのサーバータイプを指定できます。

  - **PRIMARY** PRIMARY タイプのミラーサーバーは、実際のデータベースサーバーではなく、仮想サーバーまたは論理サーバーを定義します。このサーバーの名前は、データベースの代替サーバー名です。アプリケーションで代替サーバー名を使用して、現在プライマリサーバーとして動作しているサーバーに接続できます。PRIMARY としてマーク付けされているサーバーも、ミラーサーバーで使用されている接続文字列を定義して、現在プライマリサーバーとして動作しているサーバーに接続します。また、スケールアウトシステムで新しいコピーノードをルートサーバーに接続する方法を定義します。データベースで使用できる PRIMARY サーバーは 1 つのみです。

  - **MIRROR** MIRROR タイプのミラーサーバーは、実際のデータベースサーバーではなく、仮想サーバーまたは論理サーバーを定義します。このサーバーの名前は、データベースの代替ミラーサーバー名です。アプリケーションで代替ミラーサーバー名を使用して、現在

読み込み専用ミラーとして動作しているサーバーに接続できます。データベースで使用できる MIRROR サーバーは 1 つのみです。

○ **ARBITER**　データベースミラーリングシステムでは、監視サーバーは、いずれの PARTNER サーバーがデータベースの所有権を取得するかを決定するのに役立ちます。監視サーバーは、パートナーサーバーが監視サーバーへの接続に使用できる接続文字列を使用して定義します。データベースで使用できる ARBITER サーバーは 1 つのみです。

○ **PARTNER**　データベースミラーリングシステムでは、PARTNER として定義されたサーバーは、プライマリサーバーとなってデータベースの所有権を取得できます。データベースミラーリングを行うには、2 つの PARTNER サーバーを定義する必要があり、両方のサーバーに接続文字列とステータスファイルが必要です。ミラーサーバー名は、-n サーバーオプションで指定されたデータベースサーバー名に対応している必要があり、connection_string ミラーサーバーオプションで指定された SERVER 接続文字列と一致している必要があります。

読み込み専用のスケールアウトシステムでは、1 つの PARTNER サーバーを定義します。このサーバーはルートサーバーであり、読み込みと書き込みの両方の処理が許可されたデータベースの唯一のコピーを実行します。

○ **COPY**　読み込み専用のスケールアウトシステムでは、この値は、データベースサーバーがコピーノードであることを指定します。このサーバー上のデータベース接続はすべて、読み込み専用です。ミラーサーバー名は、-n サーバーオプションで指定されたデータベースサーバー名に対応している必要があり、connection_string ミラーサーバーオプションで指定された SERVER 接続文字列と一致している必要があります。スケールアウトシステムにコピーノードを明示的に定義する必要はありません。接続時に、ルートノードによってコピーノードを定義できます。

● **FROM SERVER 句**　この句は、COPY タイプのミラーサーバーに対してのみ指定できます。この句は、ミラーリングシステムまたはスケールアウトシステムのサーバーのツリーを構成し、コピーノードがどのサーバーからトランザクションログページを取得するかを示します。

ミラーサーバー名または PRIMARY を使用して、親を指定できます。コピーノードの代替の親は、OR SERVER 句を使用して指定できます。

2 レベル (パートナーノードとコピーノード) のみが存在するデータベースミラーリングシステムでは、コピーノードが、現在のプライマリサーバーまたはミラーサーバーからトランザクションログページを取得します。

コピーノードは、データベースに格納されているコピーノードのミラーサーバー定義を使用して、接続先となるサーバーを判断します。コピーノードの定義から親の定義を探すことができ、親の定義から親に接続するための接続文字列を取得できます。

スケールアウトシステムにコピーノードを明示的に定義する必要はありません。接続時に、ルートノードによってコピーノードを定義できます。

● **USING AUTO PARENT 句**　この句を指定すると、このサーバーの親がプライマリサーバーによって割り当てられます。

● **ALTER PARENT FROM 句**　この句は、このミラーサーバーの親を変更し、ミラーサーバーのすべての兄弟を子に割り当てます。ALTER PARENT FROM 句で指定したサーバー名は、

このサーバーの現在の親が指定の値と一致するかどうかを確認するために使用されます。これは、すべての兄弟が同時に変更を要求した場合に、いずれか 1 つの兄弟のみが親を置き換えることができるようにするために使用されます。

- ***server-option* 句**　次のオプションがサポートされています。

  - **connection_string サーバーオプション**　サーバーに接続するための接続文字列を指定します。ユーザー ID とパスワードは必要ありません。1 つのミラーサーバーを別のミラーサーバーに接続するときにユーザー ID とパスワードは使用されないため、ミラーサーバーの接続文字列にはユーザー ID とパスワードを含めないでください。

    接続パラメーターのリストについては、「接続パラメーター」『SQL Anywhere サーバーデータベース管理』を参照してください。

  - **logfile サーバーオプション**　データベースミラーリングが使用されている場合に、ミラーサーバー間で送信されるファイル (1 行に 1 つの要求を含む) のロケーションを指定します。このファイルは、デバッグ専用です。

  - **preferred サーバーオプション**　サーバーがミラーリングシステムにおいて優先サーバーであるかどうかを指定します。YES または NO のいずれかを指定できます。優先サーバーは、可能な場合は常にプライマリサーバーとしての役割を果たします。このオプションは、PARTNER サーバーを定義する場合に指定します。

  - **state_file サーバーオプション**　ミラーリングシステムに関するステータス情報の管理に使用されるファイルのロケーションを指定します。データベースミラーリングには、このオプションは必須です。PARTNER タイプのサーバーには、ステータスファイルを指定してください。監視サーバーの場合は、サーバーの起動コマンドの一部としてロケーションが指定されます。

備考

データベースミラーリングシステムでは、ミラーサーバーのタイプに PRIMARY、MIRROR、ARBITER、または PARTNER を使用できます。

読み込み専用のスケールアウトシステムでは、ミラーサーバーのタイプに PRIMARY、PARTNER、または COPY を使用できます。

PARTNER、ARBITER、または COPY タイプのサーバーのミラーサーバー名は、ミラーリングシステムの一部であるデータベースサーバーの名前 (-n サーバーオプションで使用される名前) と一致している必要があります。これにより、各データベースサーバーで、サーバー自体の定義とその親の定義を見つけることができるようになります。

パーミッション

DBA 権限が必要です。

関連する動作

オートコミット。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「子コピーノードを追加する方法」『SQL Anywhere サーバー データベース管理』
- 「ステータス情報ファイル」『SQL Anywhere サーバー データベース管理』
- 「コピーノードへの親の自動割り当て」『SQL Anywhere サーバー データベース管理』
- 「優先データベースサーバー」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリング」『SQL Anywhere サーバー データベース管理』
- 「SYSMIRRORSERVER システムビュー」1280 ページ
- 「CREATE MIRROR SERVER 文」579 ページ
- 「COMMENT 文」512 ページ
- 「DROP MIRROR SERVER 文」719 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、scaleout_child コピーノードの親を変更し、そのノードのすべての兄弟を子に割り当てます。

```
ALTER MIRROR SERVER "scaleout_child"
AS COPY
ALTER PARENT FROM scaleout_mirror
connection_string = 'server=scaleout_child;host=winxp-2:6878';
```

ALTER PROCEDURE 文

プロシージャーを変更します。

構文 1

ALTER PROCEDURE [*owner*.]*procedure-name procedure-definition*

procedure-definition :「CREATE PROCEDURE 文」602 ページを参照してください。

構文 2

ALTER PROCEDURE [*owner*.]*procedure-name*
SET HIDDEN

構文 3

ALTER PROCEDURE [*owner*.]*procedure-name*
RECOMPILE

備考

ALTER PROCEDURE 文には、新しいプロシージャー全体を含めてください。PROC は PROCEDURE の同義語として使用できます。

- **構文 1**　ALTER PROCEDURE 文の構文は、最初の 1 語を除き、CREATE PROCEDURE 文の構文とまったく同じです。Watcom-SQL ダイアレクトと Transact-SQL ダイアレクトのプロシージャーは、いずれも ALTER PROCEDURE を使用して変更できます。

ALTER PROCEDURE では、プロシージャーの既存のパーミッションは変更されません。DROP PROCEDURE に続けて CREATE PROCEDURE を実行した場合は、実行パーミッションが再割り当てされます。

- **構文 2**　SET HIDDEN を使用して、関連するプロシージャーの定義を難読化し、解読できないようにします。このプロシージャーはアンロードして、他のデータベースに再ロードできます。

  SET HIDDEN を使用すると、デバッガーを使用したデバッグでも、プロシージャープロファイリングによっても、プロシージャー定義は表示されません。

  構文 1 と構文 2 を組み合わせることはできません。

> **注意**
> **この設定は、元に戻せません。**元のプロシージャー定義をデータベースの外部に保持することを強くおすすめします。

- **構文 3**　RECOMPILE 構文を使用して、ストアドプロシージャーを再コンパイルします。ストアドプロシージを再コンパイルすると、カタログに格納された定義が再解析され、構文が検証されます。結果セットを生成するものの RESULT 句を含まないプロシージャーの場合は、データベースサーバーがプロシージャーの結果セットの特性を判別しようとし、カタログに情報を格納します。これは、プロシージャーの作成後に、プロシージャーの参照先テーブルがカラムの追加、削除、または名前変更するように変更されている場合に役立つことがあります。

  プロシージャーの定義は、再コンパイルしても変わりません。SET HIDDEN 句で隠された定義を持つプロシージャーは再コンパイルできますが、これらの定義は隠されたままになります。

パーミッション

プロシージャーの所有者であるか、DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「ALTER FUNCTION 文」421 ページ
- 「DROP PROCEDURE 文」720 ページ
- 「プロシージャー、ファンクション、トリガー、イベント、ビューの内容を隠す」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。ALTER PROCEDURE は、SQL/2008 標準のオプションの SQL 言語機能 F381 です。ただし、SQL 標準では、ALTER PROCEDURE を使用してストアドプロシージャー定義を再定義することはできず、Transact-SQL ダイアレクトのプロシージャーも

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

サポートされていません。SQL/2008 では、SET HIDDEN または RECOMPILE はサポートされていません。

ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]

パブリケーションを変更します。Mobile Link では、パブリケーションが SQL Anywhere リモートデータベース内の同期データを識別します。SQL Remote では、統合データベース内とリモートデータベース内の両方のレプリケートされたデータがパブリケーションによって識別されます。

構文

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** }, ... **)**]

備考

この文は、Mobile Link と SQL Remote にのみ適用されます。

パブリケーションでは、「アーティクル」が 1 つのテーブルを表します。パブリケーションの変更とは、アーティクルの追加、修正、削除、またはパブリケーションの名前の変更を意味します。アーティクルを修正する場合は、そのアーティクル全体の定義を入力してください。

パブリケーションを変更する場合は、その直前にパブリケーションの同期を正常に完了させておくことをおすすめします。

FOR DOWNLOAD ONLY または WITH SCRIPTED UPLOAD として定義されているパブリケーションに WHERE 句を使用することはできません。

SUBSCRIBE BY 句は SQL Remote にのみ適用されます。

USING 句はスクリプトを使用するアップロード専用です。

Mobile Link パブリケーションのオプションは、ALTER SYNCHRONIZATION SUBSCRIPTION 文または CREATE SYNCHRONIZATION SUBSCRIPTION 文の ADD OPTION 句で設定します。

Mobile Link パブリケーションを変更する場合、START SYNCHRONIZATION SCHEMA CHANGE 文を実行した後でのみアーティクルを削除できます。

パーミッション

DBA 権限を持っているか、パブリケーションの所有者であることが必要です。文中で参照されるすべてのテーブルに対する排他アクセスが必要です。

関連する動作

オートコミット。

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」610 ページ
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」721 ページ
- SQL Anywhere Mobile Link クライアント：「パブリケーション」『Mobile Link クライアント管理』
- Ultra Light Mobile Link クライアント：「Ultra Light クライアントの同期設計」『Ultra Light データベース管理とリファレンス』
- 「パブリケーションとアーティクル」『SQL Remote』
- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」458 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」647 ページ
- 「ISYSSYNC システムテーブル」1005 ページ
- 「START SYNCHRONIZATION SCHEMA CHANGE 文 [Mobile Link]」944 ページ
- 「実行中のシステムに行ってはならない変更」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、Customers テーブルを pub_contact パブリケーションに追加します。

```
ALTER PUBLICATION pub_contact
  ADD TABLE Customers;
```

ALTER REMOTE MESSAGE TYPE 文 [SQL Remote]

特定のメッセージシステム、または作成したメッセージタイプに対する、パブリッシャーのメッセージシステム、またはパブリッシャーのアドレスを変更します。

構文

ALTER REMOTE MESSAGE TYPE *message-system*
ADDRESS *address*

message-system : **FILE** | **FTP** | **SMTP**

address : *string*

パラメーター

message-system　SQL Remote がサポートするメッセージシステムの 1 つ。次のいずれかの値を指定します。**FILE**、**FTP**、または **SMTP**。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

address 指定したメッセージシステムに対して有効なアドレスを含む文字列。

備考

この文は、指定したメッセージタイプでパブリッシャーのアドレスを変更します。

Message Agent は、サポートしているいずれかのメッセージリンクを使用して、データベースから出力メッセージを送信します。抽出ユーティリティは、リモートデータベースで GRANT CONSOLIDATE 文を実行するときにこのアドレスを使用します。

アドレスには、指定したメッセージシステムに応じた、パブリッシャーのアドレスを指定します。電子メールシステムの場合、アドレス文字列には有効な電子メールアドレスを指定します。ファイル共有システムの場合、アドレス文字列には SQLREMOTE 環境変数で指定されているディレクトリのサブフォルダーを指定します。SQLREMOTE 環境変数でディレクトリが指定されていない場合は、現在のディレクトリを指定してください。この設定は、リモートデータベースで GRANT CONSOLIDATE 文を使って無効にできます。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」613 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」776 ページ
- 「SQLREMOTE 環境変数」『SQL Anywhere サーバー データベース管理』
- 「SQL Remote メッセージシステム」『SQL Remote』
- 「メッセージタイプの変更」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、FILE メッセージリンクのパブリッシャーのアドレスを new_addr に変更します。

```
ALTER REMOTE MESSAGE TYPE file
ADDRESS 'new_addr';
```

ALTER SEQUENCE 文

シーケンスを変更します。

構文

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**]

パラメーター

RESTART WITH 句　指定したシーケンスを指定した値で再起動します。

INCREMENT BY 句　最後に割り当てられた値から次のシーケンス値までの増分量を定義します。デフォルトは 1 です。負の値を指定すると、降順のシーケンスが生成されます。INCREMENT BY 値が 0 の場合は、エラーが返されます。

MINVALUE 句　シーケンスで生成される最小値を定義します。デフォルトは 1 です。MINVALUE が (2^63-1) より大きいか -(2^63-1) より小さい場合は、エラーが返されます。また、MINVALUE が MAXVALUE より大きい場合も、エラーが返されます。

MAXVALUE 句　シーケンスで生成される最大値を定義します。デフォルトは -1 です。MAXVALUE が 2^63-1 より大きいか -(2^63-1) より小さい場合は、エラーが返されます。

CACHE 句　より速くアクセスできるようにメモリに保持される、事前に割り付けられたシーケンス値の数を指定します。キャッシュが不足すると、シーケンスキャッシュが再移植され、対応するエントリがトランザクションログに書き込まれます。キャッシュの現在の値は、チェックポイントの時点で ISYSSEQUENCE システムテーブルに転送されます。デフォルトは 100 です。

CYCLE 句　最大値または最小値に達した後に、値の生成を継続するかどうかを指定します。

備考

指定したシーケンスが見つからない場合は、エラーメッセージが返されます。

パーミッション

DBA 権限を持っているか、または、シーケンスの所有者であり、かつ RESOURCE 権限を持っている必要があります。

関連する動作

なし

参照

- 「ユニークな値を生成するためのシーケンスの使用」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE SEQUENCE 文」616 ページ
- 「DROP SEQUENCE 文」722 ページ

標準と互換性

- **SQL/2008**　ALTER SEQUENCE 文は、SQL/2008 標準のオプションの SQL 言語機能 T176 の一部です。CACHE 句はベンダー拡張です。

例

次の例は、シーケンス Test に新しい最大値を設定します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
ALTER SEQUENCE Test
  MAXVALUE 1500;
```

ALTER SERVER 文

リモートサーバーの属性を変更します。

構文

ALTER SERVER *server-name*
[**CLASS** *server-class*]
[**USING** *connection-info*]
[**CAPABILITY** *cap-name* { **ON** | **OFF** }]
[**CONNECTION CLOSE** [**CURRENT** | **ALL** | *connection-id*]]

server-class :
SAODBC
| **'ASEODBC'**
| **'DB2ODBC'**
| **'IQODBC'**
| **'MSSODBC'**
| **'ORAODBC'**
| **'MSACCESSODBC'**
| **'MYSQLODBC'**
| **'ULODBC'**
| **'ADSODBC'**
| **'ODBC'**
| **'SAJDBC'**
| **'ASEJDBC'**
| **'IQJDBC'**

connection-info :
computer-name:*port-number*[/*dbname*] | *data-source-name*

パラメーター

CLASS 句　CLASS 句は、サーバーのクラスを変更するために指定されます。

USING 句　USING 句は、サーバーの接続情報を変更するために指定されます。

CAPABILITY 句　CAPABILITY 句は、サーバーの機能の ON と OFF を切り替えます。サーバー機能の情報は ISYSCAPABILITY システムテーブルに格納されます。サーバー機能の名前には、SYSCAPABILITYNAME システムビューからアクセスできます。リモートサーバーとの最初の接続が確立されるまで、ISYSCAPABILITY システムテーブルと SYSCAPABILITYNAME システムビューにはデータが移植されません。2 回目以降の接続では、ISYSCAPABILITY システムテーブルからデータベースサーバーの機能が取得されます。

通常、サーバーの機能を変更する必要はありません。クラス ODBC の一般的なサーバーの機能を変更しなければならない場合はあります。

CONNECTION CLOSE 句　ユーザーがリモートサーバーへの接続を作成した場合、リモート接続は、ユーザーがローカルデータベースから切断するまで閉じられません。CONNECTION

CLOSE 句を使用すると、リモートサーバーへの接続を明示的に閉じることができます。この句は、リモート接続が非アクティブまたは不要になった場合に役立ちます。

次の SQL 文は同等であり、リモートサーバーへの現在の接続を閉じます。

```
ALTER SERVER server-name CONNECTION CLOSE;
```

```
ALTER SERVER server-name CONNECTION CLOSE CURRENT;
```

この構文を使用して、リモートサーバーへの ODBC 接続と JDBC 接続の両方を閉じることができます。これらの文を実行するために DBA 権限は不要です。

接続 ID を指定して特定のリモート ODBC 接続を切断すること、または ALL キーワードを指定してすべてのリモート ODBC 接続を切断することができます。接続 ID または ALL キーワードを指定して JDBC 接続を閉じようとすると、エラーが発生します。*connection-id* で指定された接続が現在のローカル接続ではない場合、接続を閉じるにはユーザーに DBA 権限が必要です。

備考

ALTER SERVER 文は、サーバーの属性を変更します。変更された設定は、次回リモートサーバーに接続する際に有効になります。

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「リモートデータアクセスのサーバークラス」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE SERVER 文」618 ページ
- 「DROP SERVER 文」723 ページ
- 「リモートデータアクセスのトラブルシューティング」『SQL Anywhere サーバー SQL の使用法』
- 「SYSCAPABILITY システムビュー」1255 ページ
- 「SYSCAPABILITYNAME システムビュー」1256 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、Adaptive Server Entreprise サーバー ase_prod のサーバークラスを変更して、SQL Anywhere に対するその接続が ODBC ベースになるようにします。このデータソース名は ase_prod です。

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_prod';
```

次の例は、サーバー infodc の機能を変更します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF;
```

次の例は、リモートサーバー rem_test への接続をすべて閉じます。

```
ALTER SERVER rem_test
CONNECTION CLOSE ALL;
```

次の例は、リモートサーバー rem_test への、接続 ID が 142536 である接続をすべて閉じます。

```
ALTER SERVER rem_test
CONNECTION CLOSE 142536;
```

ALTER SERVICE 文 [HTTP Web サービス]

既存の HTTP Web サービスを変更します。

構文

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**
| *****

パラメーター

service-name Web サービス名に使用できるのは、任意の順序の英数字文字、またはスラッシュ (/)、ハイフン (-)、アンダースコア (_)、ピリオド (.)、感嘆符 (!)、チルダ (~)、アスタリスク (*) 、アポストロフィ ('), 左カッコ (()、右カッコ ()) です。ただし、スラッシュ (/) は、サービス名の先頭または最後の文字には使用できず、2 つ以上連続 (// など) して使用することもできません。

サービス **root** に名前を付けることができますが、この名前には特別な機能があります。

TYPE 句 各サービスが特定の応答フォーマットを定義する、サービスタイプを識別します。リストされたサービスタイプのいずれかを指定します。デフォルト値はありません。

○ **'RAW'**　結果セットはフォーマットされずにクライアントに送られます。このサービスを利用するには、すべてのコンテンツのマークアップが明示的に指定されている必要があります。マークアップが付けられた現在のコンテンツを含む複雑な動的コンテンツである JavaScript およびイメージは、オンデマンドで生成できます。sa_set_http_header プロシージャーを使用して Content-Type 応答ヘッダーを設定することにより、メディアタイプを指定できます。すべてのブラウザーでマークアップを text/plaina ではなく HTML として表示するには、HTML マークアップを生成するときに Content-Type ヘッダーを 'text/html' に設定することをおすすめします。

○ **'HTML'**　結果セットはテーブルまたはビューの HTML 表現で返されます。

○ **'JSON'**　結果セットは JavaScript Object Notation (JSON) で返されます。JSON サービスは、自動的には JSON 入力を処理しません。(応答内の) データを JSON 形式で表すことのみ行います。JSON は、**application/x-www-form-urlencoded** がサポートされている POST/PUT メソッドを受け付けます。POST/PUT メソッドで **Content-Type: application/json** を指定すると、アプリケーションは http_variable('body') を使用して JSON (要求) コンテンツを取得します。SQL Anywhere は自動的には JSON 入力を解析しません。解析するかどうかはアプリケーションによって決まります。JSON の詳細については、http://www.json.org/を参照してください。

○ **'XML'**　結果セットは XML として返されます。結果セットがすでに XML の場合、それ以上フォーマットは適用されません。XML でない場合は、XML として自動的にフォーマットされます。代替手段としては、sa_set_http_header プロシージャーを使用して text/xml などの有効な Content-Type を設定した FOR XML RAW 句を使用して、RAW サービスから select を返す方法もあります。

URL 句　URL パスを受け入れるかどうか、受け入れる場合にはどのように処理するかを決定します。URL PATH を指定すると、URL と同じ機能があります。

○ **OFF**　URL 要求でサービス名に続いてパスを指定できないことを示します。OFF はデフォルト設定です。たとえば、パス 要素 /aaa/bbb/ccc のため、次の形式は許可されません。

```
http://host-name/service-name/aaa/bbb/ccc
```

Web サービスの作成時に CREATE SERVICE echo URL PATH OFF が指定されたとします。http://localhost/echo?id=1 に似た URL で次の値が生成されます。

```
HTTP_VARIABLE('id') == 1,
HTTP_HEADER('@HTTPQUERYSTRING') == id=1
```

○ **ON**　URL 要求でサービス名に続いてパスを指定できることを示します。**URL** という名前の専用の HTTP 変数を問い合わせることにより、パスの値が返されます。URL パラメーターを明示的に指定するようにサービスを定義したり、HTTP_VARIABLE 関数を使用して取得したりできます。たとえば、次の形式は許可されます。

```
http://host-name/service-name/aaa/bbb/ccc
```

Web サービスの作成時に CREATE SERVICE echo URL PATH ON が指定されたとします。http://localhost/echo/one/two?id=1 に似た URL で次の値が生成されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
HTTP_VARIABLE('id') == 1,
HTTP_VARIABLE('URL') == one/two,
HTTP_HEADER('@HTTPQUERYSTRING') == id=1
```

○ **ELEMENTS**　URL 要求でサービス名に続いてパスを指定できることを示します。**URL1**、**URL2** などの単一のパラメーターキーワードを指定することにより、セグメントでパスが取得されます。各パラメーターは、HTTP_VARIABLE 関数または NEXT_HTTP_VARIABLE 関数を使用して検索できます。これらの繰り返し関数は、可変の数のパス要素を指定できるアプリケーションで使用できます。たとえば、次の形式は許可されます。

```
http://host-name/service-name/aaa/bbb/ccc
```

Web サービスの作成時に CREATE SERVICE echo URL PATH ELEMENTS が指定されたとします。http://localhost/echo/one/two?id=1 に似た URL で次の値が生成されます。

```
HTTP_VARIABLE('id') == 1,
HTTP_VARIABLE('URL1') == one,
HTTP_VARIABLE('URL2') == two,
HTTP_HEADER('@HTTPQUERYSTRING') == id=1
```

10 個までの要素を取得できます。対応する要素が指定されていない場合は、NULL 値が返されます。上記の例では、対応する要素が指定されていないため、HTTP_VARIABLE('URL3') は NULL を返します。

URL の詳細については、「HTTP Web サーバーの参照」『SQL Anywhere サーバー プログラミング』と「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

AUTHORIZATION 句　サービスに接続するときに、ユーザーが基本的な HTTP 認証を通じてユーザー名とパスワードを指定する必要があるかどうかを決定します。デフォルト値は ON です。AUTHORIZATION が OFF の場合、すべてのサービスに AS 句が必要となり、USER 句でユーザーを指定する必要があります。そのユーザーのアカウントとパーミッションを使用して、すべての要求が実行されます。AUTHORIZATION が ON の場合、すべてのユーザーはユーザー名とパスワードを指定する必要があります。オプションとして、USER 句でユーザー名またはグループ名を指定することにより、サービスを使用できるユーザーを制限することもできます。ユーザー名が NULL の場合、既知のユーザーはすべてサービスにアクセスできます。AUTHORIZATION 句を使用すると、Web サービスでデータベース認証とパーミッションを使用して、データベース内のデータへのアクセスを制御できます。

authorization の値が ON の場合、Web サービスに接続しようとしている HTTP クライアントは、基本認証 (RFC 2617) を使用します。基本認証では、ユーザーとパスワードの情報が base-64 エンコーディングを使用して難読化されます。セキュリティを強化するため、HTTPS プロトコルを使用することをおすすめします。

ENABLE 句と DISABLE 句　サービスを使用できるかどうかを指定します。デフォルトでは、サービスの作成時に有効にされます。サービスを作成または変更するときに、ENABLE 句または DISABLE 句を含めることができます。サービスを無効にすると、事実上、サービスがオフラインになります。無効にしたサービスは、後で ENABLE 句を含む ALTER SERVICE 文を使用して有効にできます。無効にされたサービスに対して HTTP 要求を行うと、通常、404 Not Found HTTP ステータスが返されます。

METHODS 句　サービスでサポートされている HTTP メソッドを指定します。有効な値は、DEFAULT、POST、GET、HEAD、PUT、DELETE、NONE です。アスタリスク (*) は、POST、

GET、HEAD メソッドを表す省略形として使用できます。これらのメソッドは、RAW、HTML、XML サービスタイプのデフォルトの要求タイプです。すべての HTTP メソッドがすべてのサービスタイプに有効なわけではありません。次の表は、各サービスタイプに適用できる有効な HTTP メソッドの概要を示します。

メソッド値	適用対象のサービス	説明
DEFAULT	すべて	DEFAULT は、指定されたサービスタイプのデフォルトの HTTP メソッドのセットをリセットする場合に使用します。他のメソッド値が存在するリストに含めることはできません。
POST	RAW、HTML、JSON、XML	デフォルトでは有効になっています。
GET	RAW、HTML、JSON、XML	デフォルトでは有効になっています。
HEAD	RAW、HTML、JSON、XML	デフォルトでは有効になっています。
PUT	RAW、HTML、JSON、XML	デフォルトでは有効になっていません。
DELETE	RAW、HTML、JSON、XML	デフォルトでは有効になっていません。
NONE	すべて	NONE は、サービスへのアクセスを無効にするために使用します。
*	RAW、HTML、JSON、XML	**'POST,GET,HEAD'** と指定することと同じです。

たとえば、次のいずれかの句を使用して、サービスですべての HTTP メソッドタイプがサポートされていることを指定できます。

```
METHODS '*,PUT,DELETE'
METHODS 'POST,GET,HEAD,PUT,DELETE'
```

サービスタイプに対する要求タイプのリストをデフォルトにリセットするには、次の句を使用できます。

```
METHODS 'DEFAULT'
```

SECURE 句　セキュアリスナーまたは非セキュアリスナーでサービスにアクセスできるかどうかを指定します。ON は、HTTPS 接続のみが受け入れられ、HTTP ポートが受信する接続は HTTPS ポートに自動的にリダイレクトされることを示します。OFF は、Web サーバーの起動時に必要なポートが指定されていると、HTTP 接続と HTTPS 接続の両方が受け入れられることを示します。デフォルト値は OFF です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

USER 句　Web サービス要求を実行するパーミッションを持つデータベースユーザーまたはユーザーのグループを指定します。サービスが AUTHORIZATION OFF で設定されている場合は、USER 句を指定する必要があり、AUTHORIZATION ON で指定する必要があります。(デフォルト) 認証を必要とするサービスに対して HTTP 要求を行うと、401 Authorization Required HTTP 応答ステータスになります。この応答に基づいて、ユーザー ID とパスワードを要求するプロンプトが Web ブラウザーで表示されます。

> **警告**
> 認証が有効になっている場合 (デフォルト) には、USER 句を指定することを強くおすすめします。指定しないと、すべてのユーザーが認証されます。

USER 句は、サービス要求を処理するのに使用できるデータベースユーザーアカウントを制御します。データベースのアクセスパーミッションは、サービスのユーザーに割り当てられた権限に制限されます。

statement　サービスにアクセスしたときに起動する、ストアドプロシージャー呼び出しなどのコマンドを指定します。

statement なしの DISH 以外のサービスに対する HTTP 要求により、URL 内で実行する SQL 式が指定されます。認証が必要ですが、この機能はサーバーを SQL の挿入に対して脆弱にするため、実稼動システムでは使用しないでください。サービス内で文が定義されると、指定された SQL 文はサービスによって実行できる文のみとなります。

通常の Web サービスアプリケーションでは、*statement* を使用して関数またはプロシージャーを呼び出します。ホスト変数をパラメーターとして渡して、クライアント提供の HTTP 変数にアクセスできます。

次の *statement* は、**AuthenticateUser** という名前のプロシージャーに 2 つのホスト変数を渡すプロシージャー コールを示しています。この呼び出しは Web クライアントが **user_name** および **user_password** 変数を供給することを仮定しています。

```
CALL AuthenticateUser ( :user_name, :user_password );
```

関数またはプロシージャーに渡すホスト変数の詳細については、「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

備考

ALTER SERVICE 文は、Web サービスの属性を変更します。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「CREATE SERVICE 文 [HTTP Web サービス]」622 ページ
- 「DROP SERVICE 文」724 ページ
- 「チュートリアル：SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』
- 「SYSWEBSERVICE システムビュー」1333 ページ
- 「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「sa_set_http_header システムプロシージャー」1185 ページ
- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「root Web サービスの作成とカスタマイズ」『SQL Anywhere サーバー プログラミング』
- 「HTTP Web サーバーでの Web サービスアプリケーションの開発」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、ALTER SERVICE 文を使用して既存の Web サービスを無効にする方法を示します。

```
CREATE SERVICE WebServiceTable
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT *
    FROM SYS.SYSTAB;

ALTER SERVICE WebServiceTable DISABLE;
```

ALTER SERVICE 文 [SOAP Web サービス]

HTTP または DISH サービス経由の既存の SOAP を変更します。

構文 1 - HTTP を介した SOAP サービス

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**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| **HEAD**
| **NONE**

構文 2 - DISH サービス

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**
| *****

パラメーター

ALTER SERVICE の句の説明は、CREATE SERVICE 文の場合と同じです。

パラメーター

service-name Web サービス名に使用できるのは、任意の順序の英数字文字、またはスラッシュ (/)、ハイフン (-)、アンダースコア (_)、ピリオド (.)、感嘆符 (!)、チルダ (~)、アスタリスク (*) 、アポストロフィ (')、左カッコ (()、右カッコ ()) です。ただし、スラッシュ (/) は、サービス名の先頭または最後の文字には使用できず、2 つ以上連続 (// など) して使用することもできません。

他のサービスとは異なり、スラッシュ (/) は、DISH サービス名のどこにも使用できません。

サービス **root** に名前を付けることができますが、この名前には特別な機能があります。

TYPE 句 各サービスが特定の応答フォーマットを定義する、サービスタイプを識別します。リストされたサービスタイプのいずれかを指定します。デフォルト値はありません。

○ **'SOAP'** 結果セットは、SOAP エンベロープと呼ばれる XML ペイロードとして返されます。FORMAT 句を使用すると、データのフォーマットをさらに絞り込むことができます。SOAP サービスへの要求は、単なる汎用 HTTP 要求ではなく有効な SOAP 要求である必要があります。SOAP 標準の詳細については、http://www.w3.org/TR/2000/NOTE-SOAP-20000508/を参照してください。

○ **'DISH'** DISH サービス (SOAP ハンドラーを決定) は、GROUP コンテキスト内で SOAP サービスを参照する SOAP 終了ポイントです。また、SOAP クライアントツールキットで使用するために WSDL (Web Services Description Language) を生成することにより、SOAP サービスに対してインターフェイスも公開します。

GROUP 句　GROUP 句が指定されていない DISH サービスは、データベース内で定義されているすべての SOAP サービスを公開します。慣例により、SOAP サービス名は、GROUP 要素と NAME 要素で構成できます。最後のスラッシュ文字によって、名前がグループから区切られます。たとえば、**'aaa/bbb/ccc'** と定義されている SOAP サービス名は **'ccc'** であり、グループは **'aaa/bbb'** です。この方法で DISH サービスを区切るのは、無効です。代わりに、SOAP 終了ポイントとなる SOAP サービスのグループを指定するために、GROUP 句が適用されます。

注意
有効な XML を生成するため、WSDL ではスラッシュがアンダースコアに変換されます。GROUP 句が指定されていない DISH サービスでは、スラッシュを含む可能性のあるすべての SOAP サービスが公開されるため、このような DISH サービスを使用する場合には注意してください。アンダースコアを含む SOAP サービス名とグループを一緒に使用する場合は、あいまいにならないように注意してください。

DATATYPE 句　SOAP サービスにだけ適用されます。DATATYPE OFF が指定されていると、SOAP の入力パラメーターと応答データは XMLSchema 文字列型として定義されます。ほとんどの場合、実際のデータ型を使用すると、SOAP クライアントで計算を行う前にデータをキャストする必要がなくなるため、この方法をおすすめします。パラメーターのデータ型は、DISH サービスが生成する WSDL のスキーマセクションで公開されます。出力データ型は、データのカラムごとに XML スキーマ型の属性として表されます。

DATATYPE 句に指定できる値を次に示します。

○ **ON**　入力パラメーターと結果セットの応答のデータ型指定を生成します。

○ **OFF**　すべての入力パラメーターと応答のデータ型は、**XMLSchema** 文字列 (デフォルト) になります。

○ **IN**　入力パラメーターに対してのみ実際のデータ型を生成します。応答のデータ型は **XMLSchema** 文字列になります。

○ **OUT**　応答に対してのみ実際のデータ型を生成します。入力パラメーターは **XMLSchema** 文字列になります。

SOAP サービスの詳細については、「チュートリアル: SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

XMLSchema 型と SQL データ型とのマッピングの詳細については、「SOAP のデータ型」『SQL Anywhere サーバー プログラミング』を参照してください。

FORMAT 句　この句は、SOAP クライアントアプリケーションに応答を送信するときの出力フォーマットを指定します。

SOAP サービスフォーマットが SOAP サービスで指定されていない場合は、関連する DISH サービスフォーマットによって指定されます。デフォルトのフォーマットは DNET です。

SOAP サービスのグループにフォーマットルールを活用するには、SOAP 要求を DISH サービス (SQL Anywhere SOAP 終了ポイント) に指示してください (SOAP 操作)。SOAP サービスの FORMAT 指定は、DISH サービスの指定よりも優先されます。DISH サービスのフォーマット指

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

定は、SOAP サービスで FORMAT 句が定義されていない場合に使用されます。どちらのサービスにも FORMAT が指定されていない場合のデフォルトは **'DNET'** です。

次のフォーマットがサポートされます。

○ **'DNET'**　.NET クライアントアプリケーションで使用するために、互換性のある System.Data.DataSet フォーマットで出力されます (デフォルト)。

○ **'CONCRETE'**　この出力フォーマットは、ローとカラムのオブジェクトの配列を表すインターフェイスを生成できるが、DNET フォーマットを使用できないクライアント SOAP ツールキットをサポートするために使用されます。Java クライアントおよび .NET クライアントは、この出力フォーマットを簡単に使用できます。

DISH サービスの WSDL では、特定の出力フォーマットが公開されます。CONCRETE OFF または最後の手段として、1 つ以上の SOAP サービスの CONCRETE フォーマットが **SimpleDataset** で示されます。WSDL を調べると、任意の数のカラムの配列で構成されるローの配列で、**SimpleDataset** が構成されていることがわかります。特定のカラム名およびデータ型は指定されていないため、この表現方法は理想的ではありません。SOAP サービスで、順に RESULT 句を定義するストアドプロシージャーを呼び出すことをお勧めします。このようにして定義された SOAP サービスを公開する DISH サービスは、WSDL を生成するときに結果セットを完全に記述します。

デフォルトでは、**EXPLICIT ON** が使用され、SOAP サービスの結果セットが記述できる場合は、SOAP サービスごとに特定の Dataset エントリが WSDL に含まれます。各エントリ名の先頭には、SOAP サービス名とアンダースコアが付けられます。たとえば、**test** という名前の SOAP サービスは、各カラム要素の XMLSchema 定義が含まれた **test_Dataset** オブジェクト指定を生成します。

EXPLICIT ON が指定されている場合 (デフォルト)、次の条件が満たされると、WSDL は明示的な **DataSet** 要素を記述します。

●CREATE SERVICE 文がストアドプロシージャーを呼び出している

●カラムとデータ型を記述する RESULT 句がストアドプロシージャーで指定されている

EXPLICIT OFF が指定されている場合、WSDL は **SimpleDataset** 要素を記述します。この説明では、カラム数、カラム名またはデータ型は提供されません。

○ **'XML'**　出力は XMLSchema 文字列フォーマットで生成されます。応答としての XML ドキュメントでは、カラムデータを抽出するために SOAP クライアントでさらに処理が必要です。このフォーマットは、ローおよびカラムの配列を表す中間インターフェイスオブジェクトを生成できない SOAP クライアントに適しています。

○ **NULL**　NULL タイプを指定すると、SOAP サービスまたは DISH サービスでデフォルトの動作が使用されます。ALTER SERVICE 文で NULL タイプを使用すると、既存のサービスのフォーマットタイプが上書きされます。

AUTHORIZATION 句　サービスに接続するときに、ユーザーが基本的な HTTP 認証を通じてユーザー名とパスワードを指定する必要があるかどうかを決定します。デフォルト値は ON です。AUTHORIZATION が OFF の場合、SOAP サービスに AS 句が必要となり、USER 句でユー

ザーを指定する必要があります。そのユーザーのアカウントとパーミッションを使用して、すべての要求が実行されます。AUTHORIZATION が ON の場合、すべてのユーザーはユーザー名とパスワードを指定する必要があります。オプションとして、USER 句でユーザー名またはグループ名を指定することにより、サービスを使用できるユーザーを制限することもできます。ユーザー名が NULL の場合、既知のユーザーはすべてサービスにアクセスできます。
AUTHORIZATION 句を使用すると、Web サービスでデータベース認証とパーミッションを使用して、データベース内のデータへのアクセスを制御できます。

authorization の値が ON の場合、Web サービスに接続しようとしている HTTP クライアントは、基本認証 (RFC 2617) を使用します。基本認証では、ユーザーとパスワードの情報が base-64 エンコーディングを使用して難読化されます。セキュリティを強化するため、HTTPS プロトコルを使用することをおすすめします。

ENABLE 句と DISABLE 句　サービスを使用できるかどうかを指定します。デフォルトでは、サービスの作成時に有効にされます。サービスを作成または変更するときに、ENABLE 句または DISABLE 句を含めることができます。サービスを無効にすると、事実上、サービスがオフラインになります。無効にしたサービスは、後で ENABLE 句を含む ALTER SERVICE 文を使用して有効にできます。無効にされたサービスに対して HTTP 要求を行うと、通常、404 Not Found HTTP ステータスが返されます。

METHODS 句　サービスでサポートされている HTTP メソッドを指定します。有効な値は、DEFAULT、POST、GET、HEAD、NONE です。POST、GET、HEAD メソッドを表す省略形として、アスタリスク (*) を使用できます。SOAP サービスのデフォルトのメソッドタイプは、POST と HEAD です。DISH サービスのデフォルトのメソッドタイプは、GET、POST、および HEAD です。すべての HTTP メソッドがすべてのサービスタイプに有効なわけではありません。次の表は、各サービスタイプに適用できる有効な HTTP メソッドの概要を示します。

メソッド値	適用対象のサービス	説明
DEFAULT	両方	DEFAULT は、指定されたサービスタイプのデフォルトの HTTP メソッドのセットをリセットする場合に使用します。他のメソッド値が存在するリストに含めることはできません。
POST	両方	SOAP ではデフォルトで有効になります。
GET	DISH のみ	DISH ではデフォルトで有効になります。
HEAD	両方	SOAP と DISH でデフォルトで有効になります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メソッド値	適用対象のサービス	説明
NONE	両方	NONE は、サービスへのアクセスを無効にするために使用します。SOAP サービスに適用すると、SOAP 要求がそのサービスに直接アクセスできなくなります。このため、DISH サービスの SOAP 終了ポイント経由で SOAP 操作にアクセスします。 各 SOAP サービスに METHOD **NONE** を指定することをおすすめします。
*	DISH のみ	**'POST,GET,HEAD'** と指定することと同じです。

たとえば、次のいずれかの句を使用して、サービスですべての HTTP を介した SOAP メソッドタイプがサポートされていることを指定できます。

```
METHODS 'POST,HEAD'
```

サービスタイプに対する要求タイプのリストをデフォルトにリセットするには、次の句を使用できます。

```
METHODS 'DEFAULT'
```

SECURE 句　セキュアリスナーまたは非セキュアリスナーでサービスにアクセスできるかどうかを指定します。ON は、HTTPS 接続のみが受け入れられ、HTTP ポートが受信する接続は HTTPS ポートに自動的にリダイレクトされることを示します。OFF は、Web サーバーの起動時に必要なポートが指定されていると、HTTP 接続と HTTPS 接続の両方が受け入れられることを示します。デフォルト値は OFF です。

USER 句　Web サービス要求を実行するパーミッションを持つデータベースユーザーまたはユーザーのグループを指定します。サービスが AUTHORIZATION OFF で設定されている場合は、USER 句を指定する必要があり、AUTHORIZATION ON で指定する必要があります (デフォルト)。認証を必要とするサービスに対して HTTP 要求を行うと、401 Authorization Required HTTP 応答ステータスになります。この応答に基づいて、ユーザー ID とパスワードを要求するプロンプトが Web ブラウザーで表示されます。

警告
認証が有効になっている場合 (デフォルト) には、USER 句を指定することを強くおすすめします。指定しないと、すべてのユーザーが認証されます。

USER 句は、サービス要求を処理するのに使用できるデータベースユーザーアカウントを制御します。データベースのアクセスパーミッションは、サービスのユーザーに割り当てられた権限に制限されます。

statement　サービスにアクセスしたときに起動する、ストアドプロシージャー呼び出しなどのコマンドを指定します。

DISH サービスは、NULL 文または文のいずれかを指定する必要がある唯一のサービスです。SOAP サービスで分を定義する必要があります。他のサービスが NULL 文を持つ場合もありますが、AUTHORIZATION ON で設定した場合だけです。

statement なしの DISH 以外のサービスに対する HTTP 要求により、URL 内で実行する SQL 式が指定されます。認証が必要ですが、この機能はサーバーを SQL の挿入に対して脆弱にするため、実稼動システムでは使用しないでください。サービス内で文が定義されると、指定された SQL 文はサービスによって実行できる文のみとなります。

通常の Web サービスアプリケーションでは、*statement* を使用して関数またはプロシージャーを呼び出します。ホスト変数をパラメーターとして渡して、クライアント提供の HTTP 変数にアクセスできます。

次の *statement* は、**AuthenticateUser** という名前のプロシージャーに 2 つのホスト変数を渡すプロシージャー コールを示しています。この呼び出しは Web クライアントが **user_name** および **user_password** 変数を供給することを仮定しています。

```
CALL AuthenticateUser ( :user_name, :user_password );
```

関数またはプロシージャーに渡すホスト変数の詳細については、「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

備考

ALTER SERVICE 文は、Web サービスの属性を変更します。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「CREATE SERVICE 文 [SOAP Web サービス]」628 ページ
- 「DROP SERVICE 文」724 ページ
- 「チュートリアル：SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』
- 「SYSWEBSERVICE システムビュー」1333 ページ
- 「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「root Web サービスの作成とカスタマイズ」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、ALTER SERVICE 文を使用して既存の Web サービスを無効にする方法を示します。

```
CREATE SERVICE WebServiceTable
  TYPE 'SOAP'
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    AUTHORIZATION OFF
    USER DBA
    AS SELECT *
      FROM SYS.SYSTAB;

ALTER SERVICE WebServiceTable DISABLE;
```

ALTER SPATIAL REFERENCE SYSTEM 文

既存の空間参照系の設定を変更します。空間参照系を変更する前に、注意事項について「備考」の項を参照してください。

構文

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** }
| **POLYGON FORMAT** *polygon-format*
| **STORAGE FORMAT** *storage-format*

srs-id : *整数*

semi-major-axis-length : *数値*

semi-minor-axis-length : *数値*

inverse-flattening-ratio : *数値*

grid-size : *DOUBLE* : 通常は 0 ～ 1

tolerance-distance : *数値*

axis-order : { **'x/y/z/m'** | **'long/lat/z/m'** | **'lat/long/z/m'** }

polygon-format : { **'CounterClockWise'** | **'Clockwise'** | **'EvenOdd'** }

storage-format : { **'Internal'** | **'Original'** | **'Mixed'** }

パラメーター

各句の詳細な定義については、CREATE SPATIAL REFERENCE SYSTEM 文を参照してください。

IDENTIFIED BY 句　この句は、空間参照系の SRID 番号を変更するときに使用します。

DEFINITION 句　この句は、座標系のデフォルトを設定または上書きするときに使用します。

ORGANIZATION 句　この句は、新しい空間参照系の基となる空間参照系を作成した組織に関する情報を指定するときに使用します。

TRANSFORM DEFINITION 句　この句は、空間参照系に使用する変換の説明を指定するときに使用します。現時点では、PROJ.4 変換のみがサポートされています。

変換定義は、空間参照系間でデータを変換するときに ST_Transform メソッドで使用されます。一部の変換には、*transform-definition-string* が定義されていなくてもかまいません。

COORDINATE 句　この句は、空間参照系の次元に境界を指定するときに使用します。*coordinate-name* は、空間参照系で使用される座標系の名前です。非地理的座標系の場合は、*coordinate-name* に x、y、または m を使用できます。地理的座標系の場合は、*coordinate-name* に LATITUDE、LONGITUDE、z、または m を使用できます。

LINEAR UNIT OF MEASURE 句　この句は、空間参照系の線形測定単位を指定するときに使用します。指定する値は、ST_UNITS_OF_MEASURE システムビューで定義された線形測定単位と一致している必要があります。

ANGULAR UNIT OF MEASURE 句　この句は、空間参照系の角度測定単位を指定するときに使用します。指定する値は、ST_UNITS_OF_MEASURE システムテーブルで定義された角度測定単位と一致している必要があります。

TYPE 句　この TYPE 句は、空間参照系で点と点を結ぶ線を解釈する方法を制御するために使用します。地理的空間参照系では、TYPE 句で ROUND EARTH (デフォルト) または PLANAR のいずれかを指定できます。非地理的空間参照系の場合は、タイプに PLANAR を指定します。

ELLIPSOID 句　ELLIPSOID 句は、タイプ ROUND EARTH の空間参照系の楕円として、地球を表現するときに使用する値を指定するために使用します。DEFINITION 句が指定されている場合は、この句で楕円定義を指定できます。ELLIPSOID 句が指定されていると、このデフォルトの楕円が上書きされます。

SNAP TO GRID 句　平らな地球 (平面) の空間参照系では、SNAP TO GRID 句を使用して、SQL Anywhere で計算に使用されるグリッドサイズを定義します。SNAP TO GRID DEFAULT を指定すると、グリッドサイズがデータベースサーバーで使用されるデフォルトに設定されます。

曲面の空間参照系では、SNAP TO GRID を 0 に設定してください。

TOLERANCE 句　平らな地球 (平面) の空間参照系では、TOLERANCE 句を使用して、点を比較するときに使用する精度を指定します。

曲面の空間参照系では、TOLERANCE を 0 に設定してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

POLYGON FORMAT 句　POLYGON FORMAT 句は、多角形の解釈を変更するために使用します。次の値がサポートされています。

○ 'CounterClockwise'
○ 'Clockwise'
○ 'EvenOdd'

デフォルトの多角形フォーマットは 'EvenOdd' です。

STORAGE FORMAT 句　STORAGE FORMAT 句は、空間データをデータベースにロードしたときに格納される内容を制御するために使用します。可能な値は、次のとおりです。

○ **'Internal'**　SQL Anywhere は正規化された表現のみを格納します。元の入力特性を再現する必要がない場合に指定します。これは、平面 (TYPE PLANAR) の空間参照系のデフォルトです。

> **注意**
> Mobile Link を使用して空間データを同期する場合は、代わりに 「Mixed」 を指定してください。Mobile Link では同期時に等価性がテストされるため、データが元のフォーマットである必要があります。

○ **'Original'**　SQL Anywhere は元の表現のみを格納します。元の入力特性を再現できますが、格納された値のすべての操作に対して正規化の手順を繰り返す必要があり、データの操作が遅くなる可能性があります。

○ **'Mixed'**　SQL Anywhere で内部バージョンが格納されます。元のバージョンと異なる場合は、元のバージョンも格納されます。両方のバージョンを格納すると、元の表現特性を再現でき、格納された値の操作に対して正規化の手順を繰り返す必要はありません。ただし、ジオメトリごとに 2 つの表現が格納される可能性があるため、記憶領域の要件が大幅に増加します。

Mixed は、曲面の空間参照系 (TYPE ROUND EARTH) のデフォルトフォーマットです。

備考

空間参照系で参照している既存のデータがある場合、空間参照系を変更することはできません。たとえば、ST_Point(SRID=8743) として宣言したカラムがある場合、SRID 8743 を持つ空間参照系は変更できません。これは、格納フォーマットなどの空間参照系の多くの属性が、データの格納フォーマットに影響を及ぼすためです。SRID を参照するデータがある場合は、新しい空間参照系を作成して、データを新しい SRID に変換します。

パーミッション

DBA 権限を持つか、SYS_SPATIAL_ADMIN_ROLE グループのメンバーである必要があります。

関連する動作

なし

参照

- 「CREATE SPATIAL REFERENCE SYSTEM 文」635 ページ
- 「DROP SPATIAL REFERENCE SYSTEM 文」724 ページ
- 「空間データの使用」『SQL Anywhere サーバー 空間データサポート』
- ST_Geometry タイプの ST_Transform メソッド『SQL Anywhere サーバー 空間データサポート』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、mySpatialRef という架空の空間参照系の多角形フォーマットを EvenOdd に変更します。

```
ALTER SPATIAL REFERENCE SYSTEM mySpatialRef
POLYGON FORMAT 'EvenOdd';
```

ALTER STATISTICS 文

テーブルの 1 つのカラムまたは複数のカラムに関する統計情報を自動的に更新するかどうかを制御します。

構文

ALTER STATISTICS
[**ON**] *table* [(*column1* [, *column2* ...])]
AUTO UPDATE { **ENABLE** | **DISABLE** }

パラメーター

ON　ワード ON はオプションです。ON を指定しても、文の実行には影響がありません。

AUTO UPDATE 句　カラムの統計情報の自動更新を有効にするか無効にするかを指定します。

備考

クエリ、DML 文、LOAD TABLE 文の通常実行時に、データベースサーバーはオプティマイザーが使用するカラムの統計情報を自動的に維持します。一部のカラムでは、統計情報を維持するときに、生成に必要なオーバーヘッドに見合う利点がない場合もあります。たとえば、カラムのクエリ頻度が低い場合、または定期的に大規模な変更があっても最終的にロールバックする場合、継続的に統計情報を更新する意味はほとんどありません。ALTER STATISTICS 文は、このようなカラムの統計情報の自動更新を抑制するときに使用します。

自動更新が無効の場合でも、CREATE STATISTICS 文と DROP STATISTICS 文を使用して、カラムの統計情報を更新できます。パフォーマンスが改善されると判断した場合にのみ、更新してください。通常、カラムの統計情報は無効にしません。

パーミッション

DBA 権限

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

自動更新を無効にすると、統計情報は古くなります。改めて有効にしてもすぐに最新状態には更新されません。必要に応じて統計情報を再作成するには、CREATE STATISTICS 文を実行します。

参照

- 「CREATE STATISTICS 文」643 ページ
- 「DROP STATISTICS 文」727 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、Customers テーブルの Street カラムに関する統計情報の自動更新を無効にします。

```
ALTER STATISTICS Customers ( Street ) AUTO UPDATE DISABLE;
```

ALTER SYNCHRONIZATION PROFILE 文 [Mobile Link]

SQL Anywhere の同期プロファイルを変更します。同期プロファイルは、同期の制御に使用できる同期オプションの名前付きコレクションです。

構文

ALTER SYNCHRONIZATION PROFILE *name*
MERGE *string*

パラメーター

name　変更する同期プロファイルの名前。

MERGE 句　この句を使用して、既存のオプションを変更したり、同期プロファイルに新しいオプションを追加したりします。

string　1 つ以上の同期オプションの値ペアがセミコロンで区切られた文字列。たとえば、'option1=value1;option2=value2' のように記述します。

備考

同期プロファイルによって、SQL Anywhere データベースが Mobile Link サーバーと同期する方法を定義します。

ALTER SYNCHRONIZATION PROFILE 文に MERGE が使用されている場合は、同期プロファイルにすでに存在するオプションに文字列に指定されたオプションが追加されます。プロファイルに文字列のオプションがすでに存在する場合は、プロファイルにすでに格納済みの値が文字列の値に置き換わります。

たとえば、次の文を実行すると、プロファイル *myProfile* の値は、*subscription=s2;verbosity=high;uploadonly=on* になります。

```
CREATE SYNCHRONIZATION PROFILE myProfile 'subscription=p1;verbosity=high';
ALTER SYNCHRONIZATION PROFILE myProfile MERGE 'subscription=p2;uploadonly=on';
```

拡張オプションの設定には、次の構文を使用します。

```
ALTER SYNCHRONIZATION PROFILE myprofile MERGE
's=mysub;e={ctp=tcpip;adr="host=localhost;port=2439"}';
```

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link]」646 ページ
- 「DROP SYNCHRONIZATION PROFILE 文 [Mobile Link]」729 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]

SQL Anywhere リモートデータベースで同期サブスクリプションのプロパティを変更します。

構文

ALTER SYNCHRONIZATION SUBSCRIPTION
{ *subscription-name* | **TO** *publication-name* [**FOR** *ml-username*, ...] }

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**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

protocol-options : *string*

value : *string* | *integer*

option : *identifier*

script-version : *string*

パラメーター

TO 句　この句は、パブリケーション名を指定します。

FOR 句を指定しないで TO 句を使用した場合、RENAME 句または SET SCRIPT VERSION 句は使用できません。

FOR 句　この句は、1 つ以上の Mobile Link ユーザー名を指定します。

FOR 句を省略すると、パブリケーションに対するプロトコルタイプ、プロトコルオプション、拡張オプションが設定されます。

FOR 句を指定しないで TO 句を使用した場合、RENAME 句または SET SCRIPT VERSION 句は使用できません。

RENAME 句　この句は、サブスクリプションの新しい名前を指定します。

FOR 句を指定しないで TO 句を使用した場合、RENAME 句は使用できません。

TYPE 句　同期に使用するネットワークプロトコルを指定します。デフォルトのプロトコルは tcpip です。

ADDRESS 句　Mobile Link サーバーのロケーションを含むネットワークプロトコルオプションを指定します。

「クラウドに関する注意：」クラウドテナントデータベースの *protocol-option* 値に **NULL** を指定できます。

ADD OPTION、ALTER OPTION、DELETE OPTION、DELETE ALL OPTION 句　拡張オプションの追加、修正、削除、すべての削除ができます。それぞれの句に、オプションは 1 つのみ指定できます。Delete All にオプションは指定しません。

各オプションの値に、"=", ",", ";" の記号は使用できません。

SET SCRIPT VERSION 句　この句は、同期中に使用するスクリプトバージョンを指定します。スキーマを変更しないでスクリプトバージョンを変更できます。

FOR 句を指定しないで TO 句を使用した場合、SET SCRIPT VERSION 句は使用できません。

「クラウドに関する注意：」クラウドテナントデータベースの *script-version* 値に **NULL** を指定できます。

備考

network-protocol、*protocol-options*、*options* は、複数の個所で設定できます。

この文を使用すると、オプションや他の情報が SQL Anywhere の ISYSSYNC システムテーブルに格納されます。データベースの DBA 権限を持つユーザーであれば、パスワードや暗号化の証明書などの情報を表示できます。このとき考えられるセキュリティ上の問題を回避するために、dbmlsync コマンドラインに関する情報を指定できます。

パーミッション

DBA 権限と、パブリケーションで参照されるすべてのテーブルへの排他アクセス。

関連する動作

オートコミット。

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」610 ページ
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」721 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」647 ページ
- SQL Anywhere Mobile Link クライアント：「同期サブスクリプションの作成」『Mobile Link クライアント管理』
- Ultra Light Mobile Link クライアント：「Ultra Light クライアントの同期設計」『Ultra Light データベース管理とリファレンス』
- 「ISYSSYNC システムテーブル」1005 ページ
- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』
- 「Mobile Link SQL Anywhere クライアントの拡張オプション」『Mobile Link クライアント管理』
- 「CommunicationType (ctp) 拡張オプション」『Mobile Link クライアント管理』
- 「dbmlsync 構文」『Mobile Link クライアント管理』
- 「優先順位」『Mobile Link クライアント管理』
- 「スクリプトバージョン」『Mobile Link サーバー管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、販売サブスクリプションの Mobile Link サーバーのアドレスを変更します。

```
ALTER SYNCHRONIZATION SUBSCRIPTION sales
TYPE TCPIP
ADDRESS 'host=10.11.12.132;port=2439';
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ALTER SYNCHRONIZATION USER 文 [Mobile Link]

SQL Anywhere リモートデータベースで Mobile Link ユーザーのプロパティを変更します。

構文

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**

protocol-options : *string*

value : *string* | *integer*

パラメーター

TYPE 句　同期に使用するネットワークプロトコルを指定します。デフォルトのプロトコルは tcpip です。

ADDRESS 句　*protocol-options* を *keyword*=*value* の形式でセミコロンで区切って指定します。どのような設定を指定するかは、使用する通信プロトコル (TCP/IP、TLS、HTTP、HTTPS) に応じて異なります。

「クラウドに関する注意：」クラウドテナントデータベースの *protocol-option* 値に **NULL** を指定できます。

ADD OPTION、ALTER OPTION、DELETE OPTION、DELETE ALL OPTION 句　拡張オプションの追加、修正、削除、すべての削除ができます。それぞれの句に、オプションは 1 つしか指定できません。Delete All にオプションは指定しません。

備考

network-protocol、*protocol-options*、*options* は、複数の個所で設定できます。

この文を使用すると、オプションや他の情報が SQL Anywhere の ISYSSYNC システムテーブルに格納されます。データベースの DBA 権限を持つユーザーであれば、パスワードや暗号化の証明書などの情報を表示できます。このとき考えられるセキュリティ上の問題を回避するために、dbmlsync コマンドラインに関する情報を指定できます。

パーミッション

DBA 権限。パブリケーションで参照されるすべてのテーブルへの排他アクセスが必要です。

関連する動作

オートコミット。

参照

- 「dbmlsync 構文」『Mobile Link クライアント管理』
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」650 ページ
- 「CommunicationType (ctp) 拡張オプション」『Mobile Link クライアント管理』
- 「DROP SYNCHRONIZATION USER 文 [Mobile Link]」731 ページ
- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「ISYSSYNC システムテーブル」1005 ページ
- 「Mobile Link SQL Anywhere クライアントの拡張オプション」『Mobile Link クライアント管理』
- 「優先順位」『Mobile Link クライアント管理』
- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

ALTER TABLE 文

テーブルの定義を変更したり、従属ビューを無効にしたりします。

構文 1 - 既存のテーブルの変更

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*)
| **ADD** [*constraint-name*] **CHECK** (*condition*)
| **DROP** { **DEFAULT** | **COMPUTE** | **CHECK** | **CONSTRAINT** *constraint-name* }

drop-object :
column-name
| **CHECK**
| **CONSTRAINT** *constraint-name*

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| **UNIQUE** [**CLUSTERED**] **(** *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**
| **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**] [**ASC** | **DESC**]
  | **PRIMARY KEY** [**CLUSTERED** | **NONCLUSTERED**] [**ASC** | **DESC**]
  | **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**
| **LAST USER**
| **NULL**

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**]
[**FOR OLAP WORKLOAD**]

actions :
[**ON UPDATE** *action*] [**ON DELETE** *action*]

action :
CASCADE | **SET NULL** | **SET DEFAULT** | **RESTRICT**

構文 2 - ビューの依存関係を無効化

ALTER TABLE [*owner.*]*table-name* {
DISABLE VIEW DEPENDENCIES
}

パラメーター

追加の句　次の項では、カラムやテーブルの制約をテーブルに追加するときに使用する句について説明します。

ADD *column-name* [AS] *column-data-type* [*new-column-attribute* ...] 句　この句は、テーブルに新しいカラムを追加し、カラムのデータ型と属性を指定するときに使用します。

NULL 句と NOT NULL 句　この句は、カラムに NULL を許容するかどうかを指定するときに使用します。デフォルトでは、新しいカラムに NULL 値を使用できます。BIT 型のカラムの作成時には、NOT NULL 制約が自動的に設定されますが、NULL 入力可として宣言することもできます。

DEFAULT 句　DEFAULT 値を指定する場合、カラムの値を指定しない INSERT 文のカラムの値としてこのデフォルト値が使用されます。INSERT 文はカラムの値を指定しません。DEFAULT 値を指定しない場合、これは DEFAULT NULL と同じです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DEFAULT に指定できる値を次に示します。

- ***special-value*** DEFAULT 句には複数の特別値のうちの 1 つを使用します。

- **[CURRENT] TIMESTAMP** テーブル内の各ローが最後に変更された日付を示すことができます。カラムの宣言に DEFAULT TIMESTAMP が指定されている場合は、ローを挿入するとタイムスタンプのデフォルト値が割り付けられます。この値は、ローが更新されるたびに現在の日付と時刻に基づいて更新されます。

  挿入されたローにタイムスタンプのデフォルト値を割り付け、そのローが更新されてもタイムスタンプを更新しない場合は、DEFAULT TIMESTAMP の代わりに DEFAULT CURRENT TIMESTAMP を使用します。

  DEFAULT TIMESTAMP で宣言されたカラムにはユニークな値が入ります。これにより、アプリケーションは、ほぼ同時に行われた同じローの更新を検出できます。現在の TIMESTAMP 値が直前の値と同じ場合は、default_timestamp_increment オプションの値が加えられます。

  default_timestamp_increment オプションに基づいて、SQL Anywhere の TIMESTAMP 値を自動的にトランケートできます。これは、記録されるタイムスタンプ値の精度が低い他のデータベースソフトウェアとの互換性を維持する場合に便利です。

  グローバル変数 @@dbts は、DEFAULT TIMESTAMP を使用するカラムの最後に生成された値を表す TIMESTAMP 値を返します。

- **[CURRENT] UTC TIMESTAMP** テーブル内の各ローが最後に変更された日付を示すことができます。カラムの宣言に DEFAULT UTC TIMESTAMP が指定されている場合は、ローを挿入するとタイムスタンプのデフォルト値が割り付けられます。この値は、ローが更新されるたびに現在の協定世界時 (UTC: Coordinated Universal Time) に基づいて更新されます。

  挿入されたローにタイムスタンプのデフォルト値を割り付け、そのローが更新されてもタイムスタンプを更新しない場合は、DEFAULT UTC TIMESTAMP の代わりに DEFAULT CURRENT UTC TIMESTAMP を使用します。

  このデフォルトの動作は TIMESTAMP や CURRENT TIMESTAMP と同じですが、日付と時刻が協定世界時 (UTC: Coordinated Universal Time) になる点が異なります。

- **string** 「文字列」5 ページを参照してください。

- **global-variable** 「グローバル変数」79 ページを参照してください。

- ***constant-expression*** DEFAULT 句では、データベースオブジェクトを参照していない定数式を使用できます。その結果、GETDATE や DATEADD などの関数を使用できます。式が関数または単純な値でない場合、カッコで囲みます。

- ***sequence-expression*** データベース内のシーケンスの現在の値または次の値に、DEFAULT を設定できます。

- **AUTOINCREMENT** AUTOINCREMENT を使用する場合、カラムは整数データ型の 1 つ、または真数値型にします。

テーブルに挿入する場合、AUTOINCREMENT カラムの値を指定しないと、カラム内の任意の値より大きいユニーク値が生成されます。INSERT によって、カラムの現在の最大値よりも大きなカラム値を指定すると、その値は挿入され、以降の挿入時に開始ポイントとして使用されます。

ローを削除しても AUTOINCREMENT カウンターは減りません。ローの削除によって作成されたギャップは、挿入を行うときに明示的に割り当てることによってのみ埋めることができます。最大値未満のカラム値を明示的に挿入した後、明示的に割り当てられていない次のローを、以前の最大値より 1 大きい値を使ってオートインクリメントさせます。

カラムに直前に挿入された値は、グローバル変数 @@identity を調べることによって確認できます。

AUTOINCREMENT 値は、SYSTABCOL システムビューの max_identity カラムのデータ型に応じて、符号付き 64 ビット整数として保持されます。生成された次の値が、AUTOINCREMENT が割り当てられたカラムに格納できる最大値を超えた場合は、NULL が返されます。NULL を入力できないように宣言されたカラムであると (プライマリキーのカラムである場合など)、SQL エラーが生成されます。

カラムに使用する次の値は、sa_reset_identity プロシージャーを使用してリセットできます。

- **GLOBAL AUTOINCREMENT**　このデフォルトは、Mobile Link 同期環境または SQL Remote レプリケーションにおいて複数のデータベースを使用するときのために用意されたものです。

  このオプションは AUTOINCREMENT と同じですが、ドメインはパーティションに分割されます。各分割には同じ数の値が含まれます。データベースの各コピーにユニークなグローバルデータベース ID 番号を割り当てます。SQL Anywhere では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割からのみ設定されます。

  AUTOINCREMENT キーワードの直後にカッコで分割サイズを指定できます。この分割サイズには任意の正の整数を設定できますが、通常、分割サイズは、サイズの値がすべての分割で不足しないように選択されます。

  カラムの型が BIGINT または UNSIGNED BIGINT である場合、デフォルトの分割サイズは $2^{32} = 4294967296$ です。それ以外の型のカラムの場合、デフォルトの分割サイズは $2^{16} = 65536$ です。特に、カラムの型が INT または BIGINT ではない場合は、これらのデフォルト値が適切ではないことがあるため、分割サイズを明示的に指定するのが最も賢明です。

  このデフォルトを使用する場合、各データベース内のパブリックオプション global_database_id は、ユニークな正の整数に設定します。この値は、データベースをユニークに識別し、デフォルト値の割り当て元の分割を示します。使用できる値の範囲は $np + 1$ ～ $p(n+1)$ です。ここで、n はパブリックオプション global_database_id の値を表し、p は分割サイズを表します。たとえば、分割サイズを 1000、global_database_id を 3 に設定すると、範囲は 3001 ～4000 になります。

  前の値が $p(n + 1)$ 未満であれば、このカラム内でこれまで使用した最大値より 1 大きい値が次のデフォルト値になります。カラムに値が含まれていない場合、最初のデフォルト値は $np + 1$ になります。デフォルトのカラム値は、現在の分割以外のカラムの値の影響を受けません。つまり、$np + 1$ より小さいか $p(n + 1)$ より大きい数には影響されません。Mobile Link ま

たは SQL Remote 経由で別のデータベースからレプリケートした場合、このような値になることがあります。

カラムに直前に挿入された値は、グローバル変数 @@identity を調べることによって確認できます。

GLOBAL AUTOINCREMENT 値は、SYSTABCOL システムビューの max_identity カラムのデータ型に応じて、符号付き 64 ビット整数として保持されます。NULL 値は、分割で値が不足したときにも生成されます。NULL を入力できないように宣言されたカラムであると (プライマリキーのカラムである場合など)、SQL エラーが生成されます。この場合には、別の分割からデフォルト値を選択できるように、データベースに global_database_id の新しい値を割り当ててください。未使用の値が残り少ないことを検出し、このような状態を処理するには、GlobalAutoincrement タイプのイベントを作成します。

public オプション global_database_id は、負の値に設定できないため、選択された値は常に正になります。ID 番号の最大値を制限するのは、カラムデータ型と分割サイズだけです。

public オプション global_database_id がデフォルト値の 2147483647 に設定されると、NULL 値がカラムに挿入されます。NULL 値が許可されていない場合に、ローの挿入を試みるとエラーが発生します。

カラムに使用する次の値は、sa_reset_identity プロシージャーを使用してリセットできます。

- **LAST USER**　LAST USER は、ローを最後に変更したユーザーのユーザー ID です。

  LAST USER は文字データ型のカラムでデフォルト値として使用できます。

  INSERT の場合、このデフォルトは CURRENT USER と同じ効果があります。

  UPDATE では、LAST USER のデフォルトを持つカラムが明示的に変更されていなければ、現在のユーザー名に変更されます。

  DEFAULT TIMESTAMP または DEFAULT UTC TIMESTAMP と結合して使用すると、LAST USER のデフォルトを使用して、ローを最後に変更したユーザーと日時の両方を記録できます (ただし、別々のカラムに記録されます)。

***column-constraint* 句**　この句はカラムに制約を追加するときに使用します。検査制約の例外を指定して新規の制約を追加すると、データベースサーバーは既存の値を検証して、制約を満たすことを確認します。テーブルの変更が完了した後に発生する操作の場合にのみ、検査制約が実行されます。使用できるカラムの制約を次に示します。

- **CHECK 句**　この制約で、任意の条件を検証できます。たとえば、CHECK 制約を使うと、Sex というカラムには M または F の値だけが確実に入ります。

  テーブル内の 2 つ以上のカラム間の関係 (カラム A はカラム B 未満である必要があるなど) が関与する CHECK 制約の作成が必要な場合は、代わりにテーブル制約を定義します。

- **UNIQUE 句**　この句は、カラムの値をユニークにする必要があることを指定するとき、クラスタードインデックスと非クラスタードインデックスのどちらを作成するかを指定するときに使用します。

- **PRIMARY KEY 句**　この句は、カラムをプライマリキーにするとき、クラスタードインデックスを使用するかどうかを指定するときに使用します。

- **REFERENCES 句**　この句は、別のテーブルへの参照を追加または変更するとき、一致を処理する方法を指定するとき、クラスタードインデックスを使用するかどうかを指定するときに使用します。

- **MATCH 句**　この句は、複数カラムの外部キーを使用するときに、何を一致と見なすかを制御するために使用します。また、キーの一意性を指定することで、一意性を別に宣言する必要がなくなります。

- **NULL 句と NOT NULL 句**　この句は、カラムに NULL 値を許容するかどうかを指定するときに使用します。デフォルトでは、NULL を使用できます。

COMPRESSED 句　この句はカラムを圧縮するときに使用します。

INLINE 句と PREFIX 句　INLINE 句は、ローに格納する最大 BLOB サイズをバイト単位で指定します。INLINE 句で指定された値以下のサイズの BLOB がローに格納されます。INLINE 句で指定された値を超えるサイズの BLOB は、ロー外のテーブル拡張ページに格納されます。また、BLOB のサイズが INLINE 句で指定された値より大きい場合、BLOB の先頭から数バイト分を複製してローに保持することができます。PREFIX 句は、ローに保持されるバイト数を指定します。ローの受け入れと拒否の判断に BLOB のプレフィクスバイトを必要とする要求のパフォーマンスは、PREFIX 句によって向上する可能性があります。

圧縮されたカラムのプレフィクスデータは圧縮されずに格納されるため、要求を満たすために必要なすべてのデータがプレフィクス内に格納されている場合は、解凍は必要ありません。

INLINE と PREFIX のどちらも指定しない場合、または USE DEFAULT を指定している場合、デフォルト値は次のように適用されます。

- CHAR、NCHAR、LONG VARCHAR などの文字データ型のカラムの場合、INLINE のデフォルト値は 256 で、PREFIX のデフォルト値は 8 です。

- BINARY、LONG BINARY、VARBINARY、BIT、VARBIT、LONG VARBIT、BIT VARYING、UUID など、バイナリデータ型のカラムの場合、INLINE のデフォルト値は 256 であり、PREFIX のデフォルト値は 0 です。

> **注意**
> デフォルト以外の設定が必要な特殊な環境でない限り、デフォルト値を使用するようにしてください。デフォルト値は、パフォーマンスとディスク領域の要件のバランスをとって選択されています。たとえば、INLINE に大きな値を設定し、すべての BLOB をインラインで格納するようにした場合、ローの処理パフォーマンスは低下することがあります。また、PREFIX の値を大きくしすぎると、BLOB の一部を複製したプレフィクスデータのために、BLOB の格納に必要なディスク領域のサイズが増えることになります。

値の 1 つのみを指定する場合、その他の値は指定した値と競合しない最大の値に自動的に設定されます。INLINE 値と PREFIX 値のどちらも、データベースページサイズを超えるサイズを指定できません。また、ローデータの格納に使用できないテーブルページには、小さいサイズですが予約済みのオーバーヘッドがあります。そのため、データベースのページサイズに近

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

い INLINE 値を指定すると、やや少ないバイト数がインラインで格納される可能性があります。

INDEX 句と NO INDEX 句　BLOB の格納時に (文字型またはバイナリ型のみ)、BLOB サイズの内部しきい値 (約 8 データベースページ) を超える挿入値に対して BLOB インデックスを作成する場合は、INDEX を指定します。これはデフォルトの動作です。

BLOB インデックスは、BLOB 内のランダムアクセス検索が必要なときにパフォーマンスを改善する可能性があります。ただし、ランダムアクセスの必要がない画像ファイルやマルチメディアファイルなど、BLOB 値の種類によっては、BLOB インデックスをオフにするとパフォーマンスが改善されることがあります。カラムの BLOB インデックスをオフにするには、NO INDEX を指定します。

注意
BLOB インデックスは、テーブルインデックスとは異なります。テーブルインデックスは、1 つ以上のカラムの値のインデックスとして作成されます。

IDENTITY 句　IDENTITY は、DEFAULT AUTOINCREMENT の使用に対する Transact-SQL 互換の代替手段です。SQL Anywhere では、IDENTITY を指定して定義されたカラムは DEFAULT AUTOINCREMENT として実装されます。

COMPUTE 句　カラムが COMPUTE 句を使って作成される場合、すべてのローの値は式で提供されます。この制約を付けて作成されたカラムは、読み込み専用カラムです。この値は、ローが変更されるたびにデータベースサーバーによって変更されます。COMPUTE 式は、非決定的な値を返しません。たとえば、CURRENT TIMESTAMP などの特別値や非決定的関数は指定できません。COMPUTE 式が非決定的な値を返したとしても、それをクエリに含まれる式との一致に使用することはできません。

リモートテーブルでは COMPUTE 句は無視されます。

計算カラムの値を変更しようとする UPDATE 文は、そのカラムに対応するトリガーを起動します。

ADD *table-constraint* 句　この句はテーブルの制約を追加するときに使用します。テーブルの制約によって、テーブルのデータカラムが保持できる内容が制限されます。テーブルの制約を追加または変更するときに、オプションの制約名を使用して各制約を修正または削除することができます。追加できるテーブルの制約一覧を以下に示します。

- **UNIQUE**　この句は、*column-list* に指定するカラム値をユニークにすることを指定するとき、オプションでクラスタードインデックスを使用するかどうかを指定するときに使用します。

- **PRIMARY KEY**　この句は、テーブルのプライマリキーを追加または変更するとき、クラスタードインデックスを使用するかどうかを指定するときに使用します。テーブルは CREATE TABLE 文または別の ALTER TABLE 文が作成したプライマリキーを持っていてはいけません。

- ***foreign-key***　この句は外部キーを制約として追加するときに使用します。従属マテリアライズドビューを持つテーブルに対して、ADD FOREIGN KEY 以外のサブ句を ALTER TABLE 文で使用すると、ALTER TABLE 文は失敗します。その他すべての句の場合は、従属マテリアライズドビューを無効にし、変更が完了してから、再度有効にする必要があります。

MATCH サブ句を指定すると、複数カラムの外部キーを使用するときに、何が一致と見なされるようにするかを制御できます。また、キーの一意性を指定することで、一意性を別に宣言する必要がなくなります。

ADD PCTFREE 句　各テーブルページに確保する空き領域の割合を指定します。空き領域は、データが更新されたときにローのサイズが増えた場合に使用されます。テーブルページに空き領域がない場合は、ページのローのサイズが増えるたびに、ローを複数のテーブルページに分割することが必要になり、ローの断片化が発生します。また、パフォーマンス低下の可能性があります。空き領域のパーセンテージを 0 に指定すると、各ページに空き領域を残さず、完全にパックします。空き領域のパーセントを高い値に設定すると、各ローは単独でページに挿入されます。PCTFREE が設定されない場合、または削除された場合、データベースのページサイズに応じたデフォルトの PCTFREE 値が適用されます (ページサイズが 4 KB 以上の場合は 200 バイト)。PCTFREE の値は、ISYSTAB システムテーブルに格納されます。PCTFREE を設定すると、テーブルページに対するそれ以降のすべての挿入操作で、新しい値が使用されます。しかし、すでに挿入済みであったローは影響を受けません。値を変更しなければ、そのままの値が維持されます。PCTFREE は、ベーステーブル、グローバルテンポラリテーブル、またはローカルテンポラリテーブルに対して指定できます。

変更の句　次の項では、カラムまたはテーブルの定義を変更するときに使用する句について説明します。

ALTER *column-name column-alteration* 句　この句は、指定したカラムの属性を変更するときに使用します。カラムに一意性制約、外部キー、またはプライマリキーが設定されている場合は、カラムのデフォルトのみを変更できます。ただし、その他を変更する場合は、キーまたは制約を削除してからカラムを修正してください。

***column-data-type* 句**　この句は、カラムの長さまたはデータ型を変更するときに使用します。必要に応じて、変更されるカラムのデータを新しいデータ型に変換します。変換エラーが発生すると、操作は失敗となり、テーブルは変更されません。カラムのサイズを減らすことはできません。たとえば、VARCHAR(100) を VARCHAR(50) に変更することはできません。

[NOT] NULL 句　この句は、カラムに NULL を許容するかどうかの指定を変更するときに使用します。NOT NULL を指定し、既存ローのカラム値が NULL の場合、操作は失敗し、テーブルは変更されません。

CHECK NULL　この句は、カラムの検査制約をすべて削除するときに使用します。

DEFAULT 句　この句は、カラムのデフォルト値を変更するときに使用します。

DEFAULT NULL 句　この句は、カラムのデフォルト値を削除するときに使用します。

[CONSTRAINT *constraint-name*] CHECK { NULL | (*condition*) } 句　この句はカラムに検査制約を追加するときに使用します。

テーブル内の 2 つ以上のカラム間の関係 (カラム A はカラム B 未満である必要があるなど) が関与する CHECK 制約の作成が必要な場合は、代わりにテーブル制約を定義します。

[NOT] COMPRESSED 句　この句は、カラムを圧縮するかどうかの設定を変更するときに使用します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

INLINE 句と PREFIX 句　INLINE 句は、ローに格納する最大 BLOB サイズをバイト単位で指定します。INLINE 句で指定された値以下のサイズの BLOB がローに格納されます。INLINE 句で指定された値を超えるサイズの BLOB は、ロー外のテーブル拡張ページに格納されます。また、BLOB のサイズが INLINE 句で指定された値より大きい場合、BLOB の先頭から数バイト分を複製してローに保持することができます。PREFIX 句は、ローに保持されるバイト数を指定します。ローの受け入れと拒否の判断に BLOB のプレフィクスバイトを必要とする要求のパフォーマンスは、PREFIX 句によって向上する可能性があります。

圧縮されたカラムのプレフィクスデータは圧縮されずに格納されるため、要求を満たすために必要なすべてのデータがプレフィクス内に格納されている場合は、解凍は必要ありません。

INLINE と PREFIX のどちらも指定しない場合、または USE DEFAULT を指定している場合、デフォルト値は次のように適用されます。

- CHAR、NCHAR、LONG VARCHAR などの文字データ型のカラムの場合、INLINE のデフォルト値は 256 で、PREFIX のデフォルト値は 8 です。
- BINARY、LONG BINARY、VARBINARY、BIT、VARBIT、LONG VARBIT、BIT VARYING、UUID など、バイナリデータ型のカラムの場合、INLINE のデフォルト値は 256 であり、PREFIX のデフォルト値は 0 です。

> **注意**
> デフォルト以外の設定が必要な特殊な環境でない限り、デフォルト値を使用するようにしてください。デフォルト値は、パフォーマンスとディスク領域の要件のバランスをとって選択されています。たとえば、INLINE に大きな値を設定し、すべての BLOB をインラインで格納するようにした場合、ローの処理パフォーマンスは低下することがあります。また、PREFIX の値を大きくしすぎると、BLOB の一部を複製したプレフィクスデータのために、BLOB の格納に必要なディスク領域のサイズが増えることになります。

値の 1 つのみを指定する場合、その他の値は指定した値と競合しない最大の値に自動的に設定されます。INLINE 値と PREFIX 値のどちらも、データベースページサイズを超えるサイズを指定できません。また、ローデータの格納に使用できないテーブルページには、小さいサイズですが予約済みのオーバーヘッドがあります。そのため、データベースのページサイズに近い INLINE 値を指定すると、やや少ないバイト数がインラインで格納される可能性があります。

INDEX 句と NO INDEX 句　BLOB の格納時に (文字型またはバイナリ型のみ)、BLOB サイズの内部しきい値 (約 8 データベースページ) を超える挿入値に対して BLOB インデックスを作成する場合は、INDEX を指定します。これはデフォルトの動作です。

BLOB インデックスは、BLOB 内のランダムアクセス検索が必要なときにパフォーマンスを改善する可能性があります。ただし、ランダムアクセスの必要がない画像ファイルやマルチメディアファイルなど、BLOB 値の種類によっては、BLOB インデックスをオフにするとパフォーマンスが改善されることがあります。カラムの BLOB インデックスをオフにするには、NO INDEX を指定します。

> **注意**
> BLOB インデックスは、テーブルインデックスとは異なります。テーブルインデックスは、1 つ以上のカラムの値のインデックスとして作成されます。

SET COMPUTE 句　カラムが COMPUTE 句を使って作成される場合、すべてのローの値は式で提供されます。この制約を付けて作成されたカラムは、読み込み専用カラムです。この値は、ローが変更されるたびにデータベースサーバーによって変更されます。COMPUTE 式は、非決定的な値を返しません。たとえば、CURRENT TIMESTAMP などの特別値や非決定的関数は指定できません。COMPUTE 式が非決定的な値を返したとしても、それをクエリに含まれる式との一致に使用することはできません。

リモートテーブルでは COMPUTE 句は無視されます。

計算カラムの値を変更しようとする UPDATE 文は、そのカラムに対応するトリガーを起動します。

ALTER CONSTRAINT *constraint-name* CHECK 句　この句は、指定したテーブルの検査制約を変更するときに使用します。

テーブル内の 2 つ以上のカラム間の関係 (カラム A はカラム B 未満である必要があるなど) を指定する制約を変更する場合は、代わりにテーブル制約を定義します。

削除の句　次の項では、DROP の句について説明します。

DROP DEFAULT　テーブルまたは指定したカラムに設定されたデフォルト値を削除します。既存の値は変更されません。

DROP COMPUTE　指定したカラムの COMPUTE 属性を削除します。この文はテーブル内の既存の値を変更しません。

DROP CHECK　テーブルまたは指定したカラムのすべての検査制約を削除します。DELETE CHECK も使用できます。

DROP CONSTRAINT *constraint-name*　テーブルまたは指定したカラムの指定した制約を削除します。DELETE CONSTRAINT も使用できます。

DROP *column-name*　テーブルから指定したカラムを削除します。DELETE *column-name* も使用できます。カラムがインデックス、一意性制約、外部キー、またはプライマリキーに含まれている場合は、インデックス、制約またはキーを削除してからカラムを削除してください。このようにするとカラムを参照する検査制約は削除されません。

DROP UNIQUE (*column-name* ...)　指定したカラムの一意性制約を削除します。この一意性制約を参照する外部キーがあれば、それも削除されます。DELETE UNIQUE (*column-name* ...) も使用できます。

DROP FOREIGN KEY *fkey-name*　指定した外部キーを削除します。DELETE FOREIGN KEY *fkey-name* も使用できます。

DROP PRIMARY KEY　プライマリキーを削除します。このテーブルのプライマリキーを参照するすべての外部キーも削除します。DELETE PRIMARY KEY も使用できます。

名前変更の句　次の項では、カラムまたはテーブルの定義の一部の名前変更に使用する句について説明します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

RENAME *new-table-name* テーブルの名前を *new-table-name* に変更します。場合によっては、古いテーブル名を使用しているアプリケーションを修正する必要があります。

RENAME *column-name* TO *new-column-name* カラムの名前を *new-column-name* に変更します。場合によっては、古いカラム名を使用しているアプリケーションを修正する必要があります。

RENAME CONSTRAINT *constraint-name* TO *new-constraint-name* 制約の名前を *new-constraint-name* に変更します。

ALTER TABLE...RENAME CONSTRAINT *constraint-name* TO *new-constraint-name* を RI 制約に使用すると、制約名のみが変更され、基本となるインデックス名や外部キーの役割名 (該当する場合) は変更されません。基本となるインデックス名または役割名を変更する場合は、ALTER INDEX 文を使用します。

***table-alteration* 句** この句は、次のテーブルの属性を変更するときに使用します。

PCTFREE DEFAULT この句は、テーブルの空き割合設定をデフォルト値 (ページサイズが 4 KB 以上の場合は 200 バイト) に変更するときに使用します。

[NOT] ENCRYPTED この句は、テーブルを暗号化するかどうかの設定を変更するときに使用します。テーブルを暗号化するには、データベースでテーブルの暗号化があらかじめ有効になっている必要があります。その後、データベースの作成時に指定された暗号化キーとアルゴリズムを使用してテーブルが暗号化されます。

テーブルを暗号化した後でも、暗号化前にテンポラリファイルまたはトランザクションログに含まれていたテーブルのデータはすべて暗号化されない形式で保存されます。これを解決するには、データベースを再起動してテンポラリファイルを削除します。-o オプションでバックアップユーティリティ (dbbackup) を実行するか、BACKUP 文を使用して、トランザクションログをバックアップして新規のログを開始します。

テーブルの暗号化が有効の場合、暗号化されるテーブルのテーブルページ、関連するインデックスページ、テンポラリファイルのページ、および暗号化されるテーブルのトランザクションを含むトランザクションログページが暗号化されます。

DISABLE VIEW DEPENDENCIES 句 この句は、通常の従属ビューを無効にするときにしようします。従属マテリアライズドビューは無効になりません。ALTER MATERIALIZED VIEW...DISABLE 文を実行して、それぞれの従属マテリアライズドビューを無効にします。

備考

ALTER TABLE 文は、既存テーブルのテーブル属性 (カラム定義、制約など) を変更します。

データベースサーバーは、データベース内のオブジェクトの依存関係を追跡します。テーブルのスキームを変更すると、従属ビューに影響が及ぶ場合があります。また、変更するテーブルに依存しているマテリアライズドビューがある場合、あらかじめ ALTER MATERIALIZED VIEW...DISABLE 文を使用して無効にしておきます。

ローカルテンポラリテーブル上では ALTER TABLE を使用できません。

ALTER TABLE 文は、他の接続で現在使用中のテーブルに影響を及ぼす場合は実行できません。ALTER TABLE には時間がかかり、データベースサーバーは、文の処理中にそのテーブルを参照する処理ができません。

IMMEDIATE REFRESH として定義されたテキストインデックスが構築されたカラムを変更すると、テキストインデックスがすぐに再構築されます。テキストインデックスが AUTO REFRESH または MANUAL REFRESH として定義されている場合は、テキストインデックスが次回の再表示時に再構築されます。

ALTER TABLE 文を実行すると、データベースサーバーは、自動的に再コンパイルされる従属ビューに対するカラムパーミッションをリストアしようとします。再コンパイルされたビューに存在しないカラムに対するパーミッションは、失われます。

パーミッション

次のいずれかであることが必要です。

●テーブルの所有者

●DBA 権限を持つユーザー

●テーブルに対する ALTER パーミッションを付与されたユーザー

ALTER TABLE には、テーブルへの排他的アクセスが必要です。

グローバルテンポラリテーブルは、このテンポラリテーブルを参照したすべてのユーザーが切断されるまで変更できません。

スナップショットトランザクション内では使用できません。

関連する動作

オートコミット。

チェックポイントは ALTER TABLE 操作の開始時に実行されます。また、ALTER 操作が完了するまでチェックポイントは中断されます。

カラムまたはテーブルを変更すると、変更を加えたカラムを参照するストアドプロシージャー、ビュー、その他のアイテムは動作しなくなる場合があります。

宣言されたカラムの長さまたは型を変更した場合、またはカラムを削除した場合、そのカラムの統計情報は削除されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「イベント」『SQL Anywhere サーバー データベース管理』
- 「sa_reset_identity システムプロシージャー」1162 ページ
- 「シーケンス値と AUTOINCREMENT 値からの選択」『SQL Anywhere サーバー SQL の使用法』
- 「AUTOINCREMENT カラムのあるテーブルの再ロード」『SQL Anywhere 12 変更点とアップグレード』
- 「ALTER MATERIALIZED VIEW 文」427 ページ
- 「BACKUP 文」489 ページ
- 「ALTER INDEX 文」422 ページ
- 「CREATE TABLE 文」651 ページ
- 「DROP TABLE 文」731 ページ
- MATCH 句、CREATE TABLE 文660 ページ
- 「バックアップユーティリティ (dbbackup)」『SQL Anywhere サーバー データベース管理』
- 「SQL データ型」85 ページ
- 「テーブル変更」『SQL Anywhere サーバー SQL の使用法』
- 「特殊な IDENTITY カラム」『SQL Anywhere サーバー SQL の使用法』
- 「特別値」61 ページ
- 「テーブルとカラム制約」『SQL Anywhere サーバー SQL の使用法』
- 「ユニークな値を生成するためのシーケンスの使用」『SQL Anywhere サーバー SQL の使用法』
- 「allow_nulls_by_default オプション」『SQL Anywhere サーバー データベース管理』
- 「データベース内のテーブル暗号化の有効化」『SQL Anywhere サーバー データベース管理』
- 「クラスタードインデックス」『SQL Anywhere サーバー SQL の使用法』
- 「ビューの依存性」『SQL Anywhere サーバー SQL の使用法』
- 「データベース内のテーブル暗号化の有効化」『SQL Anywhere サーバー データベース管理』
- 「カラム統計の更新」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ALTER TABLE はコア機能です。SQL/2008 標準では、ADD CONSTRAINT と DROP CONSTRAINT と同様に、ADD COLUMN と DROP COLUMN がコア機能としてサポートされています。カラムの DEFAULT 値を追加、変更、削除する ALTER [COLUMN] は、SQL 機能 F381 です。SQL/2008 でカラムのデータ型を変更するには、SQL 言語機能 F382 である SET DATA TYPE 句を指定します。ただし、SQL Anywhere では、ALTER 句を直接指定してカラムのデータ型を変更できます。

  ALTER CONSTRAINT、RENAME、PCTFREE、ENCRYPTED、DISABLE MATERIALIZED VIEW など、SQL Anywhere でサポートされている他の句はベンダー拡張です。カラム定義や、カラムとテーブルの制約定義に対する拡張機能のサポートは、SQL/2008 のベンダー拡張であるか、SQL/2008 固有のオプション機能です。

- **Transact-SQL**　ALTER TABLE は、Adaptive Server Enterprise でサポートされています。Adaptive Server Enterprise では、ADD CONSTRAINT と DROP CONSTRAINT 以外に、ADD COLUMN 句と DROP COLUMN 句がサポートされています。Adaptive Server Enterprise では、ALTER 句のキーワード ALTER ではなく MODIFY が使用されます。Adaptive Server Enterprise では、カラムの DEFAULT 値の変更に REPLACE 句が使用されます。Adaptive Server Enterprise では、特定のテーブルのトリガーを有効／無効にするときに ALTER TABLE も使用されます。この機能は、SQL Anywhere ではサポートされていません。

例

次の例は、新しいタイムスタンプカラム TimeStamp を Customers テーブルに追加します。

```
ALTER TABLE Customers
  ADD TimeStamp AS TIMESTAMP DEFAULT TIMESTAMP;
```

次の例は、前の例で追加した新しいタイムスタンプカラム TimeStamp を削除します。

```
ALTER TABLE Customers
DROP TimeStamp;
```

Customers テーブルの Street カラムは、現在 35 文字まで保持できます。最大 50 文字まで保持できるようにするには、次を実行します。

```
ALTER TABLE Customers
ALTER Street CHAR(50);
```

次の例は、Customers テーブルにカラムを追加して、各顧客に販売担当を割り当てます。

```
ALTER TABLE Customers
ADD SalesContact INTEGER
REFERENCES Employees ( EmployeeID )
ON UPDATE CASCADE
ON DELETE SET NULL;
```

この外部キーは、カスケード更新で構成され、削除される時に NULL が設定されます。従業員がその従業員 ID を変更すると、カラムが更新してこの変更を反映します。従業員が会社を辞めて、従業員 ID が削除されると、カラムは NULL に設定されます。

次の例は、SalesOrders.SalesRepresentative カラムに外部キー FK_SalesRepresentative_EmployeeID2 を作成し、Employees.EmployeeID にリンクします。

```
ALTER TABLE GROUPO.SalesOrders
  ADD CONSTRAINT FK_SalesRepresentative_EmployeeID2
  FOREIGN KEY ( SalesRepresentative )
  REFERENCES GROUPO.Employees (EmployeeID);
```

次の例では、デフォルトが AUTOINCREMENT であるカラムを追加します。この例では、カラム値を割り当てられた NULL 入力可の AUTOINCREMENT カラムを含むように既存のすべての顧客ローが変更されますが、データベースサーバーはどのローにどの値が割り当てられるかを保証しません。

```
ALTER TABLE Customers
  ADD Surrogate_key INTEGER DEFAULT AUTOINCREMENT;
```

ALTER TEXT CONFIGURATION 文

テキスト設定オブジェクトを変更します。

構文

ALTER TEXT CONFIGURATION [*owner.*]*config-name*
STOPLIST *stoplist-string*
| **DROP STOPLIST**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| { **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 : '[*operating-system*:]*library-function-name*@*library-name*[;...]'

operating-system : **UNIX**

パラメーター

STOPLIST 句　この句は、テキストインデックスの構築時に無視する単語のリストを作成したり、置き換えたりするときに使用します。このテキスト設定オブジェクトを使用すると、このリストに指定される単語もクエリで無視されます。ストップリストの単語は、スペースで区切ります。たとえば、STOPLIST 'because about therefore only' のように記述します。ストップリストの単語には、ホワイトススペースを含めることはできません。

各言語用のストップリストのサンプルは、*%SQLANYSAMP12%¥SQLAnywhere¥SQL* にあります。

ストップリストの単語に、英数字以外の文字は使用できません。ストップリストの長さは、8000 バイト未満にしてください。

単語をストップリストに加えるかどうかは、慎重に検討してください。

DROP STOPLIST 句　この句は、テキスト設定オブジェクトのストップリストを削除するときに使用します。

MINIMUM TERM LENGTH 句　NGRAM テキストインデックスを使用する場合、MINIMUM TERM LENGTH 句に指定された値は無視されます。

テキストインデックスに使用できる単語の最小文字数。テキストインデックスの構築時または再表示時に、この設定より短い単語は無視されます。このオプションの値は 0 より大きくする必要があります。このオプションを MAXIMUM TERM LENGTH より大きい値に設定すると、MAXIMUM TERM LENGTH の値が自動的に調整され、MINIMUM TERM LENGTH の新しい値と同じになります。

MAXIMUM TERM LENGTH 句　NGRAM テキストインデックスで MAXIMUM TERM LENGTH 句を使用して、文字列の分割後の N-gram のサイズを設定します。

GENERIC テキストインデックスで MAXIMUM TERM LENGTH 句を使用して、テキストインデックスに含める単語の最大文字数を設定します。テキストインデックスの構築時または再表示時に、この設定より長い単語は無視されます。MAXIMUM TERM LENGTH の値は 60 以下にする必要があります。このオプションを MINIMUM TERM LENGTH より小さい値に設定すると、MINIMUM TERM LENGTH の値が自動的に調整され、MAXIMUM TERM LENGTH の新しい値と同じになります。

TERM BREAKER 句　カラム値を単語に分割するときに使用するアルゴリズムの名前。選択肢は GENERIC (デフォルト) または NGRAM です。

○ **GENERIC**　GENERIC では、TERM BREAKER GENERIC を指定して GENERIC 単語区切りの組み込みアルゴリズムを使用するか、または、TERM BREAKER GENERIC EXTERNAL NAME 句を使用して外部アルゴリズムを指定できます。

組み込み GENERIC アルゴリズムは、英数字以外の文字で区切られた 1 つまたは複数の英数字の文字列を単語として扱います。

外部ライブラリにある単語区切りファンクションのエントリポイントを指定するには、TERM BREAKER GENERIC EXTERNAL NAME 句を指定します。このことは、インデックスの作成やクエリを行う前の単語の区切り方に関してカスタム要件 (アポストロフィを単語区切りとしてではなく、単語の一部と見なすなど) がある場合に便利です。

external-call は、複数のファンクションやライブラリを指定でき、ライブラリのファイル拡張子 (通常、Windows の場合は *.dll*、UNIX の場合は *.so*) を含めることができます。ファイル拡張子がない場合は、ライブラリに対するプラットフォーム固有のデフォルトのファイル拡張子がデータベースサーバーで使用されます。たとえば、EXTERNAL NAME 'TermBreakFunct1@myTBlib;Unix:TermBreakFunct2@myTBlib' は、Windows の場合は *myTBlib.dll* から TermBreakFunct1 ファンクションを呼び出し、UNIX の場合は *myTBlib.so* から TermBreakFunct2 ファンクションを呼び出します。

○ **NGRAM**　組み込み NGRAM アルゴリズムは文字列を N-gram に分割します。N-gram は、ある文字列中の *n* 文字分の部分文字列です。外部単語区切りが指定されていない場合、NGRAM 単語区切りは、あいまい (近似) 一致、または単語の区切りにホワイトスペースや非英数字文字を使用しないドキュメントに必要です。

PREFILTER EXTERNAL NAME 句　PREFILTER EXTERNAL NAME 句を指定して、外部ライブラリにある事前フィルターファンクションのエントリポイントを指定します。このことは、テキストデータをバイナリデータ (PDF など) から抽出する必要がある場合に便利です。また、データ (HTML など) のインデックスを作成する前に削除するフォーマット情報やイメージが、インデックスを作成するテキストに含まれている場合にも便利です。

external-call は、複数のファンクションやライブラリを指定でき、ライブラリのファイル拡張子 (通常、Windows の場合は *.dll*、UNIX の場合は *.so*) を含めることができます。ファイル拡張子がない場合は、ライブラリに対するプラットフォーム固有のデフォルトのファイル拡張子がデータベースサーバーで使用されます。たとえば、PREFILTER EXTERNAL NAME 'PrefilterFunct1@myPreFilterlib;Unix:PrefilterFunct2@myPreFilterlib' は、Windows の場合は *myPreFilterlib.dll* から PrefilterFunct1 ファンクションを呼び出し、UNIX の場合は *myPreFilterlib.so* から PrefilterFunct2 ファンクションを呼び出します。

DROP PREFILTER 句　DROP PREFILTER 句は、指定された事前フィルターライブラリの使用をテキスト設定オブジェクトに対して削除するために使用します。つまり、このテキスト設定オブジェクトを使用するインデックスをデータベースサーバーが構築するときに、事前フィルターが実行されなくなります。

SAVE OPTION VALUES 句　テキスト設定オブジェクトが作成されると、現在の date_format、time_format、timestamp_format、timestamp_with_time_zone_format の各データベースオプションには、DATE、TIME、TIMESTAMP の各カラムがテキスト設定オブジェクトにどのように保存されているかが反映されます。SAVE OPTION VALUES 句は、テキスト設定オブジェクトに保存されたオプションを値を更新して、接続に現在有効なオプションを反映するために使用します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

単語長の設定を変更する前に、インデックス化の対象とクエリ単語の解釈に対する各種設定による影響に関する説明をお読みください。

テキストインデックスは、テキスト設定オブジェクトに依存しています。この文を使用する前に、依存する AUTO テキストインデックスまたは MANUAL REFRESH テキストインデックスをトランケートし、IMMEDIATE REFRESH テキストインデックスを削除してください。

テキスト設定オブジェクトの設定を表示するには、SYSTEXTCONFIG システムビューを問い合わせます。

パーミッション

外部の事前フィルターまたは単語区切りを変更または削除するには、DBA 権限が必要です (テキスト設定オブジェクトの所有者であるのみでは十分ではありません)。

他の場合には、テキスト設定オブジェクトの所有者であるか、DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「テキスト設定オブジェクトの変更」『SQL Anywhere サーバー SQL の使用法』
- 「データベース内のテキストインデックスに関する情報の表示 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの作成または変更時の指定項目」『SQL Anywhere サーバー SQL の使用法』
- 「チュートリアル：GENERIC テキストインデックスへの全文検索の実行」『SQL Anywhere サーバー SQL の使用法』
- 「チュートリアル：あいまい全文検索の実行」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE TEXT CONFIGURATION 文」666 ページ
- 「DROP TEXT CONFIGURATION 文」732 ページ
- 「sa_char_terms システムプロシージャー」1043 ページ
- 「sa_nchar_terms システムプロシージャー」1139 ページ
- 「sa_refresh_text_indexes システムプロシージャー」1158 ページ
- 「sa_text_index_stats システムプロシージャー」1201 ページ
- 「SYSTEXTCONFIG システムビュー」1320 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、テキスト設定オブジェクト maxTerm16 を作成し、最大単語長を 16 に変更します。

```
CREATE TEXT CONFIGURATION maxTerm16 FROM default_char;
ALTER TEXT CONFIGURATION maxTerm16
  MAXIMUM TERM LENGTH 16;
```

次の文は、maxTerm16 設定オブジェクトにストップリストを追加します。

```
ALTER TEXT CONFIGURATION maxTerm16
  STOPLIST 'because about therefore only';
```

次の文は、myTextConfig テキスト設定オブジェクトの外部単語区切りを設定します。Windows と UNIX の両方のインターフェイスが指定されています。

```
ALTER TEXT CONFIGURATION myTextConfig
  TERM BREAKER GENERIC
  EXTERNAL NAME 'my_termbreaker@termbreaker.dll;Unix:my_termbreaker@libtermbreaker_r.so'
```

次の例は、myTextConfig テキスト設定オブジェクトの外部事前フィルターを設定します。Windows と UNIX の両方のインターフェイスが指定されています。

```
ALTER TEXT CONFIGURATION myTextConfig
  PREFILTER EXTERNAL NAME
'html_xml_filter@html_xml_filter.dll;UNIX:html_xml_filter@libhtml_xml_filter_r.so';
```

次の例は、myTextConfig テキスト設定オブジェクトの外部事前フィルターを削除します。

```
ALTER TEXT CONFIGURATION myTextConfig DROP PREFILTER;
```

ALTER TEXT INDEX 文

テキストインデックスの定義を変更します。

構文

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** }] }

パラメーター

RENAME 句　RENAME 句は、テキストインデックスの名前を変更するときに使用します。

REFRESH 句　REFRESH 句は、テキストインデックスの再表示タイプを設定するときに指定します。

備考

いったん作成したテキストインデックスは、IMMEDIATE REFRESH に変更したり、IMMEDIATE REFRESH から変更したりできません。いずれかの変更が必要な場合は、テキストインデックスを削除して再度作成する必要があります。

パーミッション

基本となるテーブルの所有者であるか、DBA 権限または REFERENCES パーミッションが必要です。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、この文は実行できません。

関連する動作

オートコミット。

参照

- 「CREATE TEXT INDEX 文」667 ページ
- 「ALTER TEXT INDEX 文」480 ページ
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「COMMENT 文」512 ページ
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』
- 「データベース内のテキストインデックスに関する情報の表示 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』
- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「チュートリアル：GENERIC テキストインデックスへの全文検索の実行」『SQL Anywhere サーバー SQL の使用法』
- 「チュートリアル：あいまい全文検索の実行」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

最初の文は、テキストインデックス txt_index_manual を作成し、MANUAL REFRESH として定義します。2 つ目の文は、テキストインデックスを変更して毎日自動的に再表示します。3 つ目の文は、テキストインデックスの名前を txt_index_daily に変更します。

```
CREATE TEXT INDEX txt_index_manual ON MarketingInformation ( Description )
  MANUAL REFRESH;
ALTER TEXT INDEX txt_index_manual ON MarketingInformation
  AUTO REFRESH EVERY 24 HOURS;
ALTER TEXT INDEX txt_index_manual ON MarketingInformation
  RENAME AS txt_index_daily;
```

ALTER TRIGGER 文

トリガーの定義を修正したものに置き換えます。新しいトリガーの定義全体を ALTER TRIGGER 文にインクルードします。

構文 1 - トリガー定義の変更

ALTER TRIGGER *trigger-name trigger-definition*

trigger-definition : CREATE TRIGGER 構文

構文 2 - トリガー定義の難読化

ALTER TRIGGER *trigger-name* **ON** [*owner.*] *table-name* **SET HIDDEN**

備考

- **構文 1**　ALTER TRIGGER 文の構文は、最初の 1 語を除き、CREATE TRIGGER 文の構文とまったく同じです。

  Transact-SQL 形式と Watcom-SQL 形式のどちらの CREATE TRIGGER 構文も使用できます。

- **構文 2**　SET HIDDEN を使用して、関連するトリガーの定義を難読化し、解読できないようにします。このトリガーはアンロードして、他のデータベースに再ロードできます。SET HIDDEN を使用すると、デバッガーを使用したデバッグでも、プロシージャープロファイリングによっても、トリガー定義は表示されません。

> **注意**
> SET HIDDEN 操作は元に戻せません。

パーミッション

トリガーを定義するテーブルの所有者であるか、ユーザー DBA であるか、またはテーブルに対する ALTER パーミッションと RESOURCE 権限を持っている必要があります。

関連する動作

オートコミット。

参照

- 「CREATE TRIGGER 文」670 ページ
- 「CREATE TRIGGER 文 [T-SQL]」676 ページ
- 「DROP TRIGGER 文」734 ページ
- 「プロシージャー、ファンクション、トリガー、イベント、ビューの内容を隠す」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

ALTER USER 文

ユーザー設定を変更します。

構文 1 - データベースユーザー定義の変更

ALTER USER *user-name* [**IDENTIFIED BY** *password*]
[**LOGIN POLICY** *policy-name*]
[**FORCE PASSWORD CHANGE** { **ON** | **OFF** }]

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 2 - ユーザーのログインポリシーを元の値に復元

ALTER USER *user-name*
[RESET LOGIN POLICY]

パラメーター

user-name ユーザーの名前。

IDENTIFIED BY 句 ユーザーのパスワード。パスワードのないユーザーは、データベースに接続できません。

policy-name ユーザーに割り当てるログインポリシーの名前。LOGIN POLICY 句が指定されていない場合、変更は行われません。

FORCE PASSWORD CHANGE 句 ログイン時にユーザーが新しいパスワードを指定する必要があるかどうかを制御します。この設定は、ユーザーのポリシーの password_expiry_on_next_login オプション設定を上書きします。

RESET LOGIN POLICY 句 ユーザーのログインポリシーの設定を元の値に復元します。ログインポリシーをリセットすると、max_failed_login_attempts や max_days_since_login などのログインポリシーオプションの制限を超えたためにロックされていたアカウントに、ユーザーがアクセスできるようになります。

備考

●ユーザー ID には、次に該当する値を指定できません。

○ 空白スペース、一重引用符、または二重引用符で始まる値
○ 空白スペースで終わる値
○ セミコロンを含む値

●パスワードは大文字と小文字が区別され、次に該当する値は指定できません。

○ 空白スペース、一重引用符、または二重引用符で始まる値
○ 空白スペースで終わる値
○ セミコロンを含む値
○ 256 バイト長を超える値

verify_password_function オプションは、パスワード規則 (1 桁以上の長さであることなど) の実装に使用できる関数を指定します。パスワード検証機能を使用する場合、GRANT CONNECT 文に複数のユーザー ID とパスワードを指定することはできません。

password_expiry_on_next_login 値を ON に設定すると、ユーザーが同じポリシーに割り当てられている場合でも、次回ログインすると同時にユーザーのパスワードの有効期限が切れます。ALTER USER 句と LOGIN POLICY 句を使用すると、ユーザーが次にログインしたときにパスワードの変更を強制できます。

パーミッション

どのユーザーも自分のパスワードを変更できます。他のすべての変更には、DBA 権限が必要です。

関連する動作

なし

参照

- 「ALTER LOGIN POLICY 文」424 ページ
- 「COMMENT 文」512 ページ
- 「CREATE LOGIN POLICY 文」573 ページ
- 「CREATE USER 文」677 ページ
- 「DROP LOGIN POLICY 文」716 ページ
- 「DROP USER 文」735 ページ
- 「verify_password_function オプション」『SQL Anywhere サーバー データベース管理』
- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』
- 「既存のユーザーへのログインポリシーの割り当て」『SQL Anywhere サーバー データベース管理』
- 「GRANT 文」782 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、ユーザー SQLTester を変更します。パスワードは、**welcome** に設定されます。SQLTester ユーザーには Test1 ログインポリシーが割り当てられ、次回のログイン時にパスワードの有効期限が切れることはありません。

```
ALTER USER SQLTester IDENTIFIED BY welcome
LOGIN POLICY Test1
FORCE PASSWORD CHANGE off;
```

ALTER VIEW 文

ビュー定義を修正したものに置き換えます。

構文 1 - ビュー定義の変更

ALTER VIEW
[*owner*.]*view-name* [(*column-name*, ...)] **AS** *select-statement*
[**WITH CHECK OPTION**]

構文 2 - ビューの属性の変更

ALTER VIEW
[*owner*.]*view-name* { **SET HIDDEN** | **RECOMPILE** | **DISABLE** | **ENABLE** }

パラメーター

AS 句　ビューのベースとなる SELECT 文です。SELECT 文はローカルテンポラリテーブルを参照してはなりません。また、*query-expression* には GROUP BY 句、HAVING 句、WINDOW 句、または ORDER BY 句を指定でき、UNION、EXCEPT、INTERSECT または共通テーブル式を含めることができます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

クエリのセマンティックは、クエリが ORDER BY 句を SELECT 文の TOP または FIRST 句と組み合わせないと、返されるローの順序が定義されないことを記述します。ORDER BY 句が TOP または FIRST 句のいずれかと共に指定されている場合、クエリはそれぞれ上位 *n* ロー、または 1 番目のローを返します。

WITH CHECK OPTION 句　WITH CHECK OPTION 句を指定すると、*query-expression* で定義された条件を満たさないビューへの更新や挿入が拒否されます。

SET HIDDEN 句　SET HIDDEN 句は、ビューと句の定義を難読化し、このビューを Sybase Central などのビューから見えないようにします。ビューの明示的な参照は機能します。

> **注意**
> SET HIDDEN 操作は元に戻せません。

RECOMPILE 句　RECOMPILE 句は、ビューのカラム定義を再作成するときに使用します。この句は、無効にされていないビューに使用できるという点を除き、ENABLE 句の機能と同じです。ビューが再コンパイルされると、データベースサーバーは、新しいビュー定義に指定されたカラム名に基づいて、カラムパーミッションをリストアします。カラムが再コンパイル後に存在しない場合、既存のパーミッションは失われます。

DISABLE 句　DISABLE 句は、データベースサーバーからビューを使用できないようにします。

ENABLE 句　ENABLE 句は無効にしたビューを有効にするときに使用します。ビューを有効にすると、データベースサーバーでビューのカラム定義が再作成されます。依存しているビューを有効にしてから、ビューを有効にします。

備考

1 つ以上の INSTEAD OF トリガーがあるビューで ALTER VIEW 文を実行すると、エラーが返されます。トリガーを削除してからでないと、ビューを削除または変更できません。

別のユーザーが所有するビューを変更する場合は、所有者を含めて名前を修飾する必要があります (たとえば、GROUPO.ViewSalesOrders)。名前を修飾しなかった場合、データベースサーバーは、ユーザー本人が所有する同名のビューを検索して変更します。見つからない場合は、エラーが返されます。

ビューを変更しても、ビューに対する既存のパーミッションはそのまま維持されます。このため、パーミッションの再割り当ては必要ありません。ALTER VIEW 文を使用する代わりに、DROP VIEW 文でビューを削除してから CREATE VIEW 文でビューを再作成することもできます。ただし、削除してから再作成する場合、ビューのパーミッションを再割り当てする必要があります。

構文 1 を使用してビューを完了した後に、データベースサーバーはビューを再コンパイルします。変更の種類によっては、従属ビューがあれば、データベースサーバーは従属ビューも再コンパイルしようとします。従属ビューに影響を及ぼす変更を加える場合、従属ビューの定義も変更が必要なことがあります。

警告
ビューを定義する SELECT 文にアスタリスク (*) が含まれていた場合、ビュー内のカラム数は、基本となるテーブルでカラムが追加または削除された場合に変化することがあります。ビューカラムの名前とデータ型も変化する場合があります。

- **構文 1**　この構文はビューの構造を変更するときに使用します。変更が各カラムに制限されるテーブルを変更する場合とは異なり、ビュー構造の変更には、新しい定義で全体のビュー定義を置換する必要があります。ビューを新規作成するときと同様です。

- **構文 2**　この構文は、ビュー定義を非表示にするかどうかなど、ビューの属性を変更するときに使用します。

  SET HIDDEN を使用すると、ビューをアンロードしてから他のデータベースにリロードすることができます。SET HIDDEN を使用すると、デバッガーを使用したデバッグでも、プロシージャープロファイリングによっても、ビュー定義は表示されません。非表示のビュー定義を変更するには、ビューを削除してから、CREATE VIEW 文を使用して再作成します。

  DISABLE 句を使用すると、データベースサーバーがクエリに応答するときにビューを使用できなくなります。ビューの無効化は削除と似ていますが、無効化はビュー定義がデータベースに残る点が異なります。ビューを無効にすると、従属ビューも無効になります。DISABLE 句によって従属ビューも無効になるため、無効化されるビューだけでなく、すべての従属ビューに対する排他アクセスが必要です。

パーミッション

ビューの所有者であるか、DBA 権限が必要です。

関連する動作

オートコミット。

すべてのプロシージャーとトリガーがメモリからアンロードされるため、元のビューを参照するプロシージャーやトリガーには新しいビュー定義が反映されます。ビューを頻繁に変更すると、プロシージャーとトリガーのアンロードとロードによってパフォーマンスが低下することがあります。

参照

- 「CREATE VIEW 文」681 ページ
- 「DROP VIEW 文」737 ページ
- 「CREATE MATERIALIZED VIEW 文」576 ページ
- 「ALTER MATERIALIZED VIEW 文」427 ページ
- 「プロシージャー、ファンクション、トリガー、イベント、ビューの内容を隠す」『SQL Anywhere サーバー SQL の使用法』
- 「ビューの依存性」『SQL Anywhere サーバー SQL の使用法』
- 「ビュー」『SQL Anywhere サーバー SQL の使用法』
- 「通常のビューの作成 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ATTACH TRACING 文

診断トレーシングセッションを開始します (診断テーブルへの診断情報の送信を開始します)。

構文

ATTACH TRACING TO { **LOCAL DATABASE** | *connect-string* }
[**LIMIT** { *size* | *history* }]

connect-string : データベースの接続文字列

size : **SIZE** *nnn* { **MB** | **GB** }

history : **HISTORY** *nnn* { **MINUTES** | **HOURS** | **DAYS** }

nnn : *integer*

パラメーター

- ***connect-string*** トレーシング情報を受信するデータベースに接続するときに必要な接続文字列。このパラメーターは、プロファイルされているデータベースがデータを受信するデータベースと異なる場合にのみ必要です。

  connect-string では、接続パラメーター DBF、DBKEY、DBN、Host、Server、LINKS、PWD、UID を使用できます。

  DBF は、接続するデータベースサーバーに対する相対パスで指定します。別のデータベースサーバーを指定しない場合、現在接続しているデータベースサーバーが、DBF 接続パラメーターで指定したトレーシングデータベースの起動を試みます。

  DBF パラメーターと接続パラメーター LINKS または Server を一緒に指定すると、エラーが返されます。

- **LIMIT 句** トレーシングデータベースに格納されているデータサイズの制限。サイズまたは期間で指定します。

備考

ATTACH TRACING 文は、プロファイルするデータベースのトレーシングセッションを開始するときに使用します。この文はトレーシングレベルを設定した後にのみ使用できます。Sybase Central または sa_set_tracing_level システムプロシージャーを使用してトレーシングレベルを設定できます。

セッションを開始すると、sa_diagnostic_tracing_level テーブルに設定したトレーシングレベルに応じたトレーシング情報が生成されます。LOCAL DATABASE を指定して、プロファイル対象の同じデータベース内にあるトレーシングテーブルにトレーシングデータを送信できます。または、接続文字列をそのデータベースに指定することで (*connect-string*)、トレーシングデータを別のトレーシングデータベースに送信できます。トレーシングデータベースはあらかじめ作成されている必要があります。また、トレーシングデータベースへのパーミッションも必要です。

LIMIT SIZE 句または LIMIT HISTORY 句を使用して、トレーシングデータの格納サイズを制限できます。トレーシングデータを特定のサイズに制限するときは、LIMIT SIZE 句を使用します。

単位は MB または GB です。期間を指定してトレーシングデータのサイズを制限するときは、LIMIT HISTORY 句を使用します。単位は分、時間、日です。たとえば、HISTORY 8 DAYS と指定すると、トレーシングデータベースにトレーシングデータが格納される期間は 8 日間に制限されます。

トレーシングセッションを開始するには、トレーシングデータベースと運用データベースが稼働しているデータベースサーバーで TCP/IP が実行されている必要があります。

ローカルデータベースに対するトレーシングの場合でも、機密データを含むパケットはネットワークインターフェイス上に表示されます。セキュリティ上の理由から、接続文字列に暗号化を指定できます。

現在のトレーシングレベルを参照するには、sa_diagnostic_tracing_level テーブルを参照します。

トレーシングデータの送信先を表示するには、SendingTracingTo データベースプロパティを確認します。

パーミッション

プロファイル対象のデータベースに接続する必要があります。また、DBA 権限または PROFILE 権限が必要です。

関連する動作

なし

参照

- 「DETACH TRACING 文」706 ページ
- 「REFRESH TRACING LEVEL 文」872 ページ
- 「-x dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「sa_diagnostic_tracing_level テーブル」1022 ページ
- 「sa_set_tracing_level システムプロシージャー」1191 ページ
- SendingTracingTo データベースプロパティ『SQL Anywhere サーバー データベース管理』
- 「接続パラメーター」『SQL Anywhere サーバー データベース管理』
- 「TCP/IP プロトコル」『SQL Anywhere サーバー データベース管理』
- 「診断トレーシング」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、sa_set_tracing_level システムプロシージャーを使用して、トレーシングレベルを 1 に設定します。次に、トレーシングセッションを開始します。ローカルデータベースで生成されたトレーシングデータは、別のコンピューターにある mytracingdb トレーシングデータベースに送信されます。指定した接続文字列を参照してください。トレーシングセッション中に、最長 2 時間のトレーシングデータが保持されます。ATTACH TRACING 文の例はすべて 1 行であることに注意してください。

```
CALL sa_set_tracing_level( 1 );
ATTACH TRACING TO 'uid=DBA;pwd=sql;server=remotedbsrv;dbn=mytracingdb;host=winxp-32'
 LIMIT HISTORY 2 HOURS;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

BACKUP 文

データベースとトランザクションログをバックアップします。

構文 1 - イメージのバックアップ

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** }

構文 2 - アーカイブのバックアップ

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*

パラメーター

DIRECTORY 句　バックアップファイルを作成するディスク上のロケーション。データベースサーバー起動時の現在のディレクトリを基準に指定します。このディレクトリが存在しない場合は作成されます。ディレクトリとして空の文字列を指定すると、最初にトランザクションログ

をコピーしないで名前を変更したり、トランケートしたりできます。データベースミラーリングを使用している場合は、この句を使用しないでください。

WAIT BEFORE START 句　この句は、データベースのバックアップコピー内の各接続のロールバックログが空であることを確認するときに使用します。この句を WITH CHECKPOINT LOG NO COPY 句と一緒に使用して、リカバリに必要な情報がデータベースのバックアップコピーに含まれないことを確認します。

バックアップ操作の完了に WAIT BEFORE START 句と WITH CHECKPOINT LOG NO COPY 句を使用した場合は、データベースのバックアップコピーを読み込み専用モードで開始し、検証できます。バックアップデータベースの検証を有効にすることにより、ユーザーがデータベースの追加コピーを作成することを防止できます。

この句を指定すると、アクティブなトランザクションがなくなるまでバックアップは遅延されます。データベースに対する他のすべてのアクティビティが抑制され、データベースのバックアップコピーにリカバリの必要がないことを確認するチェックポイントが実行されます。チェックポイントが完了すると、データベースに対する他のアクティビティが再開されます。

WAIT AFTER END 句　トランザクションログの名前変更やトランケートを行う場合、WAIT AFTER END 句を指定すると、すべてのトランザクションが完了してからログの名前変更やトランケートが実行されるようにできます。この句を指定した場合、バックアップは、開いているトランザクションを他の接続がコミットまたはロールバックするまで待機してから完了します。新しい入力トランザクションがバックアップを無期限に待機させることがあるため、この句は注意して使用してください。

DBFILE ONLY 句　DBFILE ONLY 句を指定すると、メインデータベースファイルと関連するすべての DB 領域のバックアップコピーが作成されますが、トランザクションログはコピーされません。DBFILE ONLY 句は、TRANSACTION LOG RENAME 句または TRANSACTION LOG TRUNCATE 句と同時に使用できません。

TRANSACTION LOG ONLY 句　TRANSACTION LOG ONLY 句を指定すると、他のデータベースファイルをコピーしないで、トランザクションログのバックアップコピーを作成できます。

TRANSACTION LOG RENAME [MATCH] 句　この句を指定すると、データベースサーバーによって、バックアップの完了時に現在のトランザクションログの名前が変更されます。MATCH キーワードを省略すると、トランザクションログのバックアップコピーの名前はデータベースの現在のトランザクションログと同じになります。MATCH キーワードを指定すると、トランザクションログのバックアップコピーの名前は、*YYMMDDnn.log* 形式になり、現在のトランザクションログのコピーの変更名と一致します。MATCH キーワードを指定すると、古いデータを上書きしないで同じ文を 2 度以上実行できます。

バックアップを完了しないでトランザクションログの名前を変更するには、TRANSACTION LOG ONLY 句で空のディレクトリ名を指定します。次に例を示します。

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

TRANSACTION LOG TRUNCATE 句　この句を指定すると、バックアップの完了時に現在のトランザクションログはトランケートされて再起動します。データベースミラーリングを使用している場合は、この句を使用しないでください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

バックアップを完了しないでトランザクションログをトランケートするには、TRANSACTION LOG ONLY 句で空のディレクトリ名を指定します。次に例を示します。

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG TRUNCATE;
```

***archive-root* 句**　アーカイブファイルのファイル名またはテープドライブデバイス名

テープ上にバックアップを作成する場合は、そのテープドライブのデバイス名を指定します。アーカイブファイル名の末尾に自動的に付加される番号は、アーカイブバックアップを実行するごとに増分します。

円記号 (¥) は、SQL 文字列のエスケープ文字であるため、2 つ重ねます。

ON EXISTING ERROR 句　この句は、イメージバックアップにだけ適用されます。デフォルトでは、既存のファイルは BACKUP DATABASE 文を実行したときに上書きされます。この句が使用されている場合は、バックアップによって作成されるファイルのいずれかがすでに存在するとエラーが発生します。

ATTENDED 句　この句は、テープデバイスにバックアップを作成する場合にのみ適用されます。ATTENDED ON (デフォルト) は、テープドライブの状況を監視し、必要に応じてテープを交換する担当者がいることを示します。テープドライブに介入が必要な場合には、BACKUP DATABASE 文を発行したアプリケーションにメッセージが送信されます。データベースサーバーはドライブが使用可能になるのを待ちます。このような状態が発生するのは、テープの交換が必要な場合などです。

ATTENDED OFF を指定すると、テープの交換が必要な場合やドライブが使用できない場合、メッセージは送信されず、エラーとなります。

WITH COMMENT 句　この句は、バックアップ履歴ファイルにコメントを記録するときに使用します。アーカイブバックアップの場合、コメントはアーカイブファイルにも記録されます。

HISTORY 句　デフォルトでは、各バックアップ操作は *backup.syb* ファイルに行を追加します。HISTORY OFF を指定して、*backup.syb* ファイルが更新されないようにすることができます。次のいずれかの条件に当てはまる場合、ファイルが更新されないようにすることがあります。

- バックアップが頻繁に行われる
- *backup.syb* ファイルを定期的にアーカイブまたは削除するプロシージャーがない
- ディスク領域が非常に限られている

AUTO TUNE WRITERS 句　バックアップが始まると、バックアップディレクトリへのバックアップファイルの書き込み専用にスレッドが 1 つ割り当てられます。ただし、バックアップディレクトリがライターの負荷の増加に対処できるデバイス上にある場合 (RAID アレイなど)、ライターとして動作するスレッド数を増やすことで全体のバックアップパフォーマンスを改善できます。この句を ON にすると (デフォルト)、データベースサーバーは、バックアップに参加しているすべてのデバイスについて、読み込みと書き込みのパフォーマンスを定期的に検証します。追加のライターを作成することで全体のバックアップ速度を改善できる場合、データベースサーバーによって追加のライターが作成されます。

WITH CHECKPOINT LOG 句　この句では、バックアップ先のディレクトリに書き込む前にバックアップでデータベースファイルに対してどのような処理を実行するかを指定します。バック

アップ中に更新前イメージを適用するか、チェックポイントログをバックアップの一部としてコピーするかを選択します。どちらを選択するかによって、パフォーマンスに違いが生じます。デフォルト設定は、イメージバックアップの場合は AUTO で、アーカイブバックアップの場合は COPY です。

○ **COPY 句**　このオプションを BACKUP 文で WAIT BEFORE START 句と併用することはできません。

COPY を指定すると、バックアップは、変更されたページを適用しないでデータベースファイルを読み込みます。チェックポイントログ全体とシステム DB 領域がバックアップディレクトリにコピーされます。このデータベースサーバーを次回起動すると、データベースは自動的にバックアップ開始時のチェックポイントの状態にリカバリされます。

このオプションを使用すると、ページをテンポラリファイルに書き込む必要がないため、バックアップパフォーマンスが向上し、バックアップ中に動作している他の接続との内部サーバー競合が減少します。ただし、チェックポイントログには変更したページの元のイメージが含まれるため、データベースが更新されるとサイズが増えます。コピーを指定すると、データベースファイルのバックアップコピーが、バックアップ開始時のデータベースファイルよりも大きくなる可能性があります。COPY オプションは、バックアップ先ディレクトリのディスク領域が十分である場合に使用してください。

○ **NO COPY 句**　NO COPY を指定すると、チェックポイントログはバックアップの一部としてコピーされません。この場合、変更されたページはテンポラリファイルに保存され、バックアップの実行中、バックアップに適用されます。データベースファイルのバックアップコピーは、バックアップを開始した時点のデータベースと同じサイズになります。

このオプションを指定すると、データベースファイルのバックアップは小さくなりますが、バックアップの速度が遅くなり、データベースサーバーで実行される他の操作のパフォーマンスが低下することがあります。バックアップ先のドライブの空き領域が少ない場合に、このオプションが役に立ちます。

○ **RECOVER 句**　RECOVER を指定すると、(COPY オプションを指定した場合と同じように) チェックポイントログがコピーされますが、このチェックポイントログはバックアップの完了時にデータベースに適用されます。したがって、データベースファイルは、バックアップ操作を開始した時点と同じ状態 (および同じサイズ) にリストアされます。このオプションが役に立つのは、バックアップドライブの空き領域が少ない場合です (チェックポイントログをバックアップする COPY オプションの場合と同じ量の空き領域が必要ですが、バックアップファイルのサイズは COPY オプションを指定した場合より小さくなります)。

○ **AUTO 句**　AUTO を指定すると、データベースサーバーはバックアップディレクトリがあるボリュームの空きディスク領域をチェックします。バックアップを開始する時点でデータベースサイズの 2 倍以上の空きディスク領域がある場合、このオプションは COPY を指定した場合と同様に動作します。それ以外の場合、NO COPY の場合と同様に動作します。AUTO がデフォルトの動作です。

MAX WRITE 句　アーカイブバックアップの場合、デフォルトで、バックアップファイルの書き込み専用にスレッドが 1 つ割り当てられます。バックアップディレクトリが、ライターの負荷の増加に対処できるデバイス (RAID アレイなど) 上にある場合、ライターとして動作するスレッド数を増やすことで全体のバックアップパフォーマンスを改善できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

AUTO を指定すると、各リーダースレッドに出力ストリームが 1 つ作成されます。値 *n* には、作成可能な出力ストリームの最大数を、リーダースレッド数を上限として指定します。この句のデフォルト値は 1 です。バックアップ先がテープの場合、使用できるライターは 1 つだけです。

最初のストリームであるストリーム 0 が *myarchive.X* という名前のファイルを生成します。*X* は 1 から始まる番号で、必要なファイルの数まで 1 ずつ増えていきます。それ以外のすべてのストリームは *myarchive.Y.Z* という名前のファイルを生成します。*Y* は 1 から始まるストリーム番号です。*Z* は 1 から始まる番号で、必要なファイル数まで 1 ずつ増えていきます。

FREE PAGE ELIMINATION 句　アーカイブバックアップでは、デフォルトで空きページがスキップされるため、バックアップが小規模で高速になる可能性があります。トランザクションログファイルには空きページがないため、空きページが削除されてもトランザクションログファイルのバックアップには効果がありません。大規模なトランザクションログファイルを使用するデータベースでは、小規模なトランザクションログファイルを使用するデータベースと比較して、空きページを削除するメリットがあまりないこともあります。

空きページの削除を有効にして強力に暗号化されているデータベースをバックアップした場合は、データベースをリストアするときに暗号化キーを指定してください。空きページの削除を無効にして強力に暗号化されているデータベースをバックアップした場合は、データベースをリストアするときに暗号化キーを指定する必要はありません。

バージョン 12 では、バージョン 11 以前のデータベースサーバーで作成したアーカイブバックアップをリストアできません。

備考

BACKUP 文は、サーバー側のバックアップを実行します。クライアント側のバックアップを実行するには、dbbackup ユーティリティを使用します。

各バックアップ操作では、イメージかアーカイブかに関係なく、履歴ファイル *backup.syb* が更新されます。backup.syb ファイルには、特定のデータベースサーバーで実行された BACKUP 操作と RESTORE 操作が記録されます。*backup.syb* ファイルのロケーションを指定する方法については、「SALOGDIR 環境変数」『SQL Anywhere サーバー データベース管理』を参照してください。

リカバリを実行しないで読み込み専用データベースサーバーで開始できるバックアップを作成するには、WAIT BEFORE START 句と WITH CHECKPOINT LOG NO COPY 句を両方とも使用する必要があります。WAIT BEFORE START 句はロールバックログを空にし、WITH CHECKPOINT LOG NO COPY 句はチェックポイントログを空にします。どちらのファイルが欠けても、リカバリが必要になります。バックアップしたデータベースをリカバリする必要がない場合は、WAIT BEFORE START 句と WITH CHECKPOINT LOG NO COPY 句の代わりに、WITH CHECKPOINT LOG RECOVER を使用できます。

- **構文 1 - イメージのバックアップ**　イメージバックアップでは、各データベースファイルのコピーが、バックアップユーティリティ (dbbackup) の場合と同じ方法で作成されます。デフォルトでは、バックアップユーティリティはクライアントコンピューターにバックアップを作成しますが、-s オプションを使用すると、バックアップユーティリティの使用時にデータベースサーバー上にバックアップを作成することもできます。一方、BACKUP DATABASE 文の場合、バックアップはデータベースサーバー上にのみ作成できます。

オプションで、データベースファイルまたはトランザクションログのどちらかだけを保存できます。また、トランザクションログは、バックアップの完了後に名前を変更するか、またはトランケートすることもできます。

または、ディレクトリとして空の文字列を指定して、ログをコピーしないで名前を変更したり、トランケートしたりできます。これは、スペースが問題となるレプリケーション環境で便利です。この機能をトランザクションログサイズのイベントハンドラーと一緒に使用すると、トランザクションログが所定のサイズに達したときに名前を変更したり、delete_old_logs オプションと一緒に使用して、トランザクションログが必要なくなったときに削除したりできます。

イメージバックアップからリストアするには、保存されたファイルを元のロケーションにコピーして、トランザクションログを再度適用します。

- **構文 2 - アーカイブのバックアップ**　アーカイブのバックアップでは、必要なバックアップ情報をすべて保持する 1 つのファイルが作成されます。送信先には、ファイル名またはテープドライブデバイス名のどちらかを指定できます。

  1 つのテープに 1 つのバックアップしか保存できません。バックアップ処理が終了すると、テープが排出されます。

  1 つのテープに 1 つのアーカイブしか作成できませんが、1 つのアーカイブを複数のテープに保存することは可能です。アーカイブのバックアップからデータベースをリストアするには、RESTORE DATABASE 文を使用します。

  RESTORE DATABASE 文が、トランザクションログのみを含むアーカイブファイルを参照している場合、リストアされるデータベースファイルのロケーションにファイルが存在しない場合でも、文ではそのファイル名を指定します。たとえば、トランザクションログのみを含むアーカイブからディレクトリ *C:¥MYNEWDB* にリストアする場合、RESTORE DATABASE 文は次のようになります。

  ```
  RESTORE DATABASE 'c:¥mynewdb¥my.db' FROM archive-root
  ```

  警告
  データベースとトランザクションログのバックアップコピーには、どのような変更でも加えるべきではありません。バックアップ中に処理中のトランザクションがなかった場合、または BACKUP DATABASE WITH CHECKPOINT LOG RECOVER か WITH CHECKPOINT LOG NO COPY を指定した場合は、読み込み専用モードを使用するか、バックアップデータベースのコピーを検証することによって、バックアップデータベースの妥当性をチェックできます。

  一方、トランザクションの処理中だった場合、または BACKUP DATABASE WITH CHECKPOINT LOG COPY を指定した場合は、検証の開始時にデータベースサーバーがデータベースのリカバリを実行する必要が生じます。リカバリを実行するとバックアップコピーに変更が加えられますが、これは望ましいことではありません。

  この文の実行中に、進行状況メッセージの表示を要求できます。

  また、Progress 接続プロパティを使用して、文がどの程度実行されたかを確認することもできます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

DBA、REMOTE DBA、または BACKUP の権限が必要です。

関連する動作

チェックポイントを発生させます。

参照

- 「トランザクションログが複数の場合のデータベースのリカバリ」『SQL Anywhere サーバー データベース管理』
- Progress 接続プロパティ『SQL Anywhere サーバー データベース管理』
- 「progress_messages オプション」『SQL Anywhere サーバー データベース管理』
- 「バックアップユーティリティ (dbbackup)」『SQL Anywhere サーバー データベース管理』
- 「イメージバックアップ」『SQL Anywhere サーバー データベース管理』
- 「文字列」5 ページ
- 「RESTORE DATABASE 文」880 ページ
- KEY 句、RESTORE DATABASE 文880 ページ
- 「バックアップとデータリカバリ」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリングとトランザクションログファイル」『SQL Anywhere サーバー データベース管理』
- 「EXECUTE IMMEDIATE 文 [SP]」740 ページ
- 「並列データベースバックアップ」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Windows Mobile**　Windows Mobile では、BACKUP DATABASE DIRECTORY 構文 (構文 1) のみがサポートされます。

例

現在のデータベースとトランザクションログをそれぞれ別のファイルにバックアップし、既存のトランザクションログ名を変更します。イメージバックアップが作成されます。

```
BACKUP DATABASE
DIRECTORY 'd:¥¥temp¥¥backup'
TRANSACTION LOG RENAME;
```

トランザクションログの名前を変更するオプションは、古いトランザクションログが引き続き必要になるレプリケーション環境で特に役立ちます。

現在のデータベースとトランザクションログをテープにバックアップします。

```
BACKUP DATABASE
TO '¥¥¥¥.¥¥tape0';
```

コピーを作成しないでトランザクションログの名前を変更します。

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

動的に構成されるディレクトリ名を指定して、BACKUP DATABASE 文を実行します。

```
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;
```

BEGIN SNAPSHOT 文

スナップショットアイソレーショントランザクションで使用するスナップショットを指定の期間に開始します。

構文

BEGIN SNAPSHOT

備考

デフォルトでは、トランザクションが始まると、アプリケーションによってテーブルの最初のローがフェッチされるまで、データベースサーバーはスナップショットの作成を遅延します。BEGIN SNAPSHOT 文を使用すると、トランザクション内で事前にスナップショットを開始できます。スナップショットトランザクションによって BEGIN SNAPSHOT 文が実行されると、データベースサーバーはスナップショットを作成します。

次の条件に一致する場合、文は失敗し、エラーを返します。

●データベースに対してスナップショットトランザクションのサポートが有効にされていない。

●現在のトランザクションに対してスナップショットがすでに開始されている。

この文は非スナップショットトランザクションにも有効です。これは、この文を使用すると、文レベルのスナップショット操作のためにトランザクションで後から使用できるスナップショットを、非スナップショットトランザクションで開始できるためです。

パーミッション

なし

関連する動作

なし

参照

- 「allow_snapshot_isolation オプション」『SQL Anywhere サーバー データベース管理』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

BEGIN 文

複合文を指定します。

構文

[*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 と *exception-declaration* :「DECLARE 文」693 ページを参照してください。

cursor-declaration :「DECLARE CURSOR 文 [ESQL] [SP]」686 ページを参照してください。

temporary-table-declaration :「DECLARE LOCAL TEMPORARY TABLE 文」691 ページを参照してください。

パラメーター

statement-label　終了の *statement-label* を指定する場合は、開始の *statement-label* と一致させます。LEAVE 文を使うと、複合文に続く最初の文から実行を再開できます。プロシージャーまたはトリガーの本文である複合文は、プロシージャーまたはトリガーの名前と同じ暗黙のラベルを持っています。

ATOMIC 句　アトミックな文とは、完全に実行されるか、またはまったく実行されない文のことです。たとえば、何千ものローを更新する UPDATE 文では、たくさんのローを更新した後にエラーが発生することがあります。文が完了しないと、すべての変更内容が元の状態に戻ります。同様に、BEGIN 文を ATOMIC と指定すると、この文は完全に実行されるか、あるいはまったく実行されないかのどちらかです。

local-declaration　BEGIN のすぐ後で、複合文は複合文の中にだけ存在するオブジェクトをローカルに宣言できます。複合文は、変数、カーソル、テンポラリテーブル、または例外に対してローカル宣言を行います。ローカル宣言は、複合文またはその中でネストされる複合文の中のどの文からでも参照できます。複合文のローカル宣言は、文の例外ハンドラーに表示されます。ローカル宣言は、複合文の中から呼び出される他のプロシージャーには表示されません。

備考

プロシージャーまたはトリガーの本文は複合文です。複合文は、プロシージャーまたはトリガー内の制御文の中でも使用できます。

複合文によって、1 つまたは複数の SQL 文をまとめて、1 つの単位として扱うことができます。複合文はキーワード BEGIN で始まり、キーワード END で終わります。

パーミッション

なし

関連する動作

なし

参照

- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』
- 「エラーと警告の処理」『SQL Anywhere サーバー SQL の使用法』
- 「例外処理とアトミックな複合文」『SQL Anywhere サーバー SQL の使用法』
- 「例外ハンドラー」『SQL Anywhere サーバー SQL の使用法』
- 「アトミックな複合文」『SQL Anywhere サーバー SQL の使用法』
- 「 CONTINUE 文」520 ページ
- 「SIGNAL 文 [SP]」934 ページ
- 「RESIGNAL 文 [SP]」879 ページ
- 「RAISERROR 文」862 ページ
- 「BEGIN 文 [TSQL]」499 ページ

標準と互換性

- **SQL/2008**　複合文を識別する BEGIN は SQL/2008 のオプションの SQL 言語機能 P002 の部分によって構成されます。

例

プロシージャーまたはトリガーの本文は複合文です。

```
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 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:
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    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;
```

下の例は、次の変数を宣言しています。

- INT、初期設定値 5 の v1。
- CHAR(10)、初期値 abc の v2 と v3。

```
BEGIN
  DECLARE v1 INT = 5
  DECLARE v2, v3 CHAR(10) = 'abc'
        // ...
END
```

BEGIN 文 [TSQL]

複合文を指定します。

構文

BEGIN
statement-list
END

statement-list :
sql-statement
| *variable-declaration*
| *cursor-declaration*
| *temporary-table-declaration*

variable-declaration : 「DECLARE 文」693 ページを参照してください。

cursor-declaration : 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページを参照してください。

temporary-table-declaration : 「DECLARE LOCAL TEMPORARY TABLE 文」691 ページを参照してください。

パラメーター

statement-list 文および宣言のリスト

備考

BEGIN 文によって、1 つまたは複数の SQL 文をまとめて、1 つの単位として扱い、キーワード BEGIN で開始して、キーワード END で終了することができます。

Transact-SQL 複合文ではエラー処理が異なります。

パーミッション

なし

関連する動作

なし

参照

- 「Transact-SQL プロシージャーでのエラー処理」『SQL Anywhere サーバー SQL の使用法』
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』
- 「BEGIN 文」497 ページ
- 「 CONTINUE 文」520 ページ
- 「GOTO 文 [T-SQL]」775 ページ
- 「RAISERROR 文」862 ページ
- 「Transact-SQL と互換性のあるデータベース」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　複合文を識別する BEGIN は SQL/2008 のオプションの SQL 言語機能 P002 の部分によって構成されます。

例

下の例は、次の変数を宣言しています。

- INT、初期設定値 5 の v1。
- CHAR(10)、初期値 abc の v2 と v3。

```
BEGIN
  DECLARE v1 INT = 5
  DECLARE v2, v3 CHAR(10) = 'abc'
        // ...
END
```

BEGIN TRANSACTION 文 [T-SQL]

ユーザー定義のトランザクションを開始します。

構文

BEGIN TRAN[SACTION] [*transaction-name*]

備考

オプションのパラメーター *transaction-name* はこのトランザクションに割り当てられた名前です。有効な識別子を指定します。トランザクション名はネストされた BEGIN/COMMIT または BEGIN/ROLLBACK 文の最も外側の組でのみ使用してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

BEGIN TRANSACTION 文をトランザクション内で実行すると、トランザクションのネストレベルが 1 つ増加します。ネストレベルは COMMIT 文で減少します。トランザクションがネストされているときは、最も外側の COMMIT だけがデータベースへの変更を保存します。

Adaptive Server Enterprise と SQL Anywhere は、両方とも 2 つのトランザクションモードを持ちます。

デフォルトの Adaptive Server Enterprise トランザクションモードは非連鎖モードと呼ばれ、明示的な BEGIN TRANSACTION 文が実行されてトランザクションを起動しないかぎり、各文を個々にコミットします。反対に、ISO SQL/2008 互換の連鎖モードは、明示的 COMMIT が実行されるときか、オートコミットする文 (データ定義文など) が実行されるときにのみトランザクションをコミットします。

chained データベースオプションを設定してモードを制御できます。ODBC 接続と SQL Anywhere の組み込みの SQL 接続のデフォルト設定が ON の場合、SQL Anywhere は連鎖モードで実行されます (ODBC を使用している場合、AutoCommit ODBC 設定も確認が必要です)。TDS 接続のデフォルトは Off です。

非連鎖モードでは、トランザクションはデータ検索やデータ修正文の前に暗黙的に開始されます。これらの文には、DELETE、INSERT、OPEN、FETCH、SELECT、UPDATE があります。トランザクションの終了は、COMMIT または ROLLBACK 文を使って明示的に行います。

トランザクション内では chained オプションの設定を変更できません。

警告
ストアドプロシージャーを呼び出すときは、目的のトランザクションモードで正しく動作することを確認してください。

現在のネストレベルはグローバル変数 @@trancount に入っています。@@trancount 変数の値は、最初の BEGIN TRANSACTION 文が実行される前は 0 であり、@@trancount の値が 1 のときに実行された COMMIT だけがデータベースへの変更を永続的なものにすることができます。

@@trancount の値は、実行された明示的な BEGIN TRANSACTION 文を数える以上の目的には使用しないでください。

Adaptive Server Enterprise が暗黙的にトランザクションを起動する場合、@@trancount 変数を 1 に設定します。トランザクションが暗黙的に起動されたとき、SQL Anywhere は @@trancount 値を 1 に設定しません。代わりに、SQL Anywhere の @@trancount 変数は、BEGIN TRANSACTION 文が始まる前は (現在のトランザクションがある場合でも) 0 の値を持ちますが、Adaptive Server Enterprise (連鎖モード) では、この変数は 1 の値を持ちます。

BEGIN TRANSACTION 文で起動するトランザクションには、最初の BEGIN TRANSACTION 文の後、@@trancount の値は SQL Anywhere と Adaptive Server Enterprise の両方で 1 です。トランザクションが異なる文で暗黙的に起動し、その後、BEGIN TRANSACTION 文が実行された場合、BEGIN TRANSACTION 文の後、@@trancount の値は、SQL Anywhere と Adaptive Server Enterprise の両方で 2 になります。

トランザクションまたはセーブポイント名のない ROLLBACK 文は、常に文を最も外側の BEGIN TRANSACTION (明示的または暗黙的) 文にロールバックし、トランザクション全体をキャンセルします。

パーミッション

なし

関連する動作

なし

参照

- 「COMMIT 文」514 ページ
- 「ROLLBACK 文」890 ページ
- 「SAVEPOINT 文」895 ページ
- 「isolation_level オプション」『SQL Anywhere サーバー データベース管理』
- 「chained オプション」『SQL Anywhere サーバー データベース管理』
- 「トランザクション内のセーブポイント」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　BEGIN TRANSACTION は Adaptive Server Enterprise によってサポートされます。

例

次のバッチは、一連の @@trancount の値を 0、1、2、1、0 と報告します。値はデータベースサーバーのメッセージウィンドウに出力されます。

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT
PRINT @@trancount
COMMIT
PRINT @@trancount
```

BREAK 文 [T-SQL]

複合文またはループから出ます。

構文

BREAK

備考

BREAK 文は、ループから出るための制御文です。実行はループの後に記述されている最初の文から再開されます。

パーミッション

なし

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

なし

参照

- 「WHILE 文 [T-SQL]」988 ページ
- 「 CONTINUE 文」520 ページ
- 「BEGIN 文」497 ページ
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** Transact-SQL 拡張。

例

この例では、BREAK 文は、最も高い製品の価格が 50 ドルを超える場合、WHILE ループをブレークします。そうでない場合、ループは平均価格が 30 ドルになるまで継続します。

```
WHILE ( SELECT AVG( UnitPrice ) FROM Products ) < $30
BEGIN
  UPDATE Products
  SET UnitPrice = UnitPrice + 2
  IF ( SELECT MAX(UnitPrice) FROM Products ) > $50
    BREAK
END
```

CALL 文

プロシージャーを呼び出します。

構文 1

[*variable* =] **CALL** *procedure-name* **(** [*expression*, ...] **)**

構文 2

[*variable* =] **CALL** *procedure-name* **(** [*parameter-name* = *expression*, ...] **)**

備考

CALL 文は、CREATE PROCEDURE 文を使ってあらかじめ作成されたプロシージャーを呼び出します。プロシージャーが完了すると、INOUT または OUT パラメーター値がコピーし直されます。

引数リストは、引数の順序を守って指定するか、キーワードフォーマットを使うことにより指定できます。順序を守って指定した場合、引数はプロシージャーのパラメーターリスト内の対応するパラメーターと一致します (オプションのパラメーターには DEFAULT を使用できます)。キーワードを使用した場合、引数は指定したパラメーターと一致します。

プロシージャー引数には、CREATE PROCEDURE 文のデフォルト値が割り当てられます。パラメーターが見つからない場合は、デフォルト値が割り当てられます。また、デフォルトが設定さ

れていない場合、パラメーターが NULL に設定されます。デフォルトが設定されていないうえ、引数も指定されていない場合には、エラーが発生します。

プロシージャーの内部では、プロシージャーが結果セットを返す場合、DECLARE 文で CALL 文を使用できます。

サブクエリと空間メソッド呼び出しは、CALL 文のストアドプロシージャーに対する引数としては使用できません。

プロシージャーは RETURN 文を使用して整数値 (たとえば、ステータスインジケーターとして) を返すことができます。変数の戻り値を保存するには、割り当て演算子として等号を使います。

```
CREATE VARIABLE returnval INT;
returnval = CALL proc_integer ( arg1 = val1, ... )
```

呼び出されたプロシージャーが INT を返し、値が NULL の場合は、代わりにエラーステータス値 0 が返されます。この場合と実際の値の 0 が返される場合とを区別する方法はありません。

注意
この文を使用した関数の呼び出しは廃止される予定です。呼び出す関数がある場合は、代入文を使用して関数を呼び出し、その結果を変数に代入することを検討してください。次に例を示します。

```
DECLARE varname INT;
SET varname=test( );
```

パーミッション

プロシージャーの所有者であるか、そのプロシージャーに対する EXECUTE パーミッションまたは DBA 権限が必要です。

関連する動作

なし

参照

- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [外部呼び出し]」550 ページ
- 「CREATE FUNCTION 文 [Web サービス]」555 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [外部呼び出し]」583 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「GRANT 文」782 ページ
- 「EXECUTE 文 [T-SQL]」744 ページ
- 「結果セット」『SQL Anywhere サーバー SQL の使用法』
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　コア機能。ストアドプロシージャーから値を返すために RETURN 文を使用するのは、ベンダー拡張です。SQL/2008 では、プロシージャーの戻り値ではなく、SQL 呼び出し

関数の戻り値のみがサポートされます。ストアドプロシージャーの引数に対するデフォルト値は、SQL/2008 ではサポートされません。

例

ShowCustomers プロシージャーを呼び出します。このプロシージャーはパラメーターがなく、結果セットを返します。

```
CALL ShowCustomers();
```

次の Interactive SQL の例は、指定された ID を持つ顧客からの注文の数を戻すプロシージャーを作成し、結果を保持する変数を作成し、プロシージャーを呼び出し、結果を表示します。

```
CREATE PROCEDURE OrderCount (IN customer_ID INT, OUT Orders INT)
BEGIN
  SELECT COUNT(SalesOrders.ID)
  INTO Orders
  FROM 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 DUMMY
go
```

CASE 文

複数の状況に基づいて実行パスを選択します。

構文 1

CASE *value-expression*
WHEN [*constant* | **NULL**] **THEN** *statement-list* ...
[**WHEN** [*constant* | **NULL**] **THEN** *statement-list*] ...
[**ELSE** *statement-list*]
END [**CASE**]

構文 2

CASE
WHEN [*search-condition* | **NULL**] **THEN** *statement-list* ...
[**WHEN** [*search-condition* | **NULL**] **THEN** *statement-list*] ...
[**ELSE** *statement-list*]
END [**CASE**]

備考

構文 1　CASE 文は制御文であり、これを使用して SQL 文のリストから式の値に対応する文を選択して実行できます。*value-expression* は、文字列、数値、日付、その他の SQL データ型など

の単一の値を取る式です。WHEN 句が *value-expression* の値に対して存在する場合、WHEN 句の中の *statement-list* が実行されます。適切な WHEN 句が存在せず、ELSE 句が存在する場合、ELSE 句の *statement-list* が実行されます。END CASE の後に記述されている最初の文から実行が再開されます。

value-expression に NULL を指定できる場合は、ISNULL 関数を使用して NULL の *value-expression* を異なる式で置き換えます。

構文 2　この形式の文は、CASE 文中で最初に条件と一致した *search-condition* に対して実行されます。条件と一致する *search-conditions* がない場合は、ELSE 句が実行されます。

式が NULL でもよい場合は、最初の *search-condition* に次の構文を使用します。

```
WHEN search-condition IS NULL THEN statement-list
```

注意
CASE 文の構文と CASE 式の構文を混同しないでください。

パーミッション

なし

関連する動作

なし

参照

- 「ISNULL 関数 [その他]」260 ページ
- 「不定の値：NULL」『SQL Anywhere サーバー SQL の使用法』
- 「BEGIN 文」497 ページ
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』
- 「CASE 式」16 ページ
- 「CASE 文 [T-SQL]」507 ページ

標準と互換性

- **SQL/2008**　CASE 文は、SQL/2008 標準の言語機能 P002 (Computational completeness) の一部です。END CASE ではなく END を単独で使用するのは、ベンダー拡張です。
- **Transact-SQL**　CASE 文は Adaptive Server Enterprise によってサポートされます。

例

CASE 文を使用する次のプロシージャーは、SQL Anywhere サンプルデータベースの Products テーブルにリストされている製品を、シャツ、帽子、ショートパンツ、不明のいずれかに分類します。

```
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
  DECLARE prod_name CHAR(20);
  SELECT Name INTO prod_name FROM Products
  WHERE ID = product_ID;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    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;
```

次の例は、構文 2 を使用して、SQL Anywhere サンプルデータベース内の製品数量に関するメッセージを生成します。

```
  CREATE PROCEDURE StockLevel (IN product_ID INT)
  BEGIN
    DECLARE qty INT;
    SELECT Quantity INTO qty FROM 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;
```

CASE 文 [T-SQL]

複数の状況に基づいて実行パスを選択します。

構文 1

CASE *value-expression*
WHEN [*constant* | **NULL**] **THEN** *statement-list* ...
[**WHEN** [*constant* | **NULL**] **THEN** *statement-list*] ...
[**ELSE** *statement-list*]
END

構文 2

CASE
WHEN [*search-condition* | **NULL**] **THEN** *statement-list* ...
[**WHEN** [*search-condition* | **NULL**] **THEN** *statement-list*] ...
[**ELSE** *statement-list*]
END

備考

構文 1　CASE 文は制御文であり、これを使用して SQL 文のリストから式の値に対応する文を選択して実行できます。*value-expression* は、文字列、数値、日付、その他の SQL データ型など

の単一の値を取る式です。WHEN 句が *value-expression* の値に対して存在する場合、WHEN 句の中の *statement-list* が実行されます。適切な WHEN 句が存在せず、ELSE 句が存在する場合、ELSE 句の *statement-list* が実行されます。END CASE の後に記述されている最初の文から実行が再開されます。

value-expression に NULL を指定できる場合は、ISNULL 関数を使用して NULL の *value-expression* を異なる式で置き換えます。

構文 2　この形式の文は、CASE 文中で最初に条件と一致した *search-condition* に対して実行されます。条件と一致する *search-conditions* がない場合は、ELSE 句が実行されます。

式が NULL でもよい場合は、最初の *search-condition* に次の構文を使用します。

```
WHEN search-condition IS NULL THEN statement-list
```

注意
CASE 文の構文と CASE 式の構文を混同しないでください。

パーミッション

なし

関連する動作

なし

参照

- 「ISNULL 関数 [その他]」260 ページ
- 「BEGIN 文」497 ページ
- 「CASE 式」16 ページ
- 「不定の値：NULL」『SQL Anywhere サーバー SQL の使用法』
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　CASE 文は、SQL/2008 標準の言語機能 P002 (Computational completeness) の一部です。SQL 標準では、CASE 文を終了するのに、END 単独ではなく END CASE が必要です。
- **Transact-SQL**　Adaptive Server Enterprise と互換性があります。

例

CASE 文を使用する次のプロシージャーは、SQL Anywhere サンプルデータベースの Products テーブルにリストされている製品を、シャツ、帽子、ショートパンツ、不明のいずれかに分類します。

```
CREATE PROCEDURE DBA.ProductType( @product_ID INTEGER,@TYPE CHAR(10) OUTPUT ) AS
BEGIN
  DECLARE @prod_name CHAR(20)
  SELECT Name INTO @prod_name FROM Products
    WHERE ID = @product_ID
  IF @prod_name
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    = '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;
```

次の例は、構文 2 を使用して、SQL Anywhere サンプルデータベース内の製品数量に関するメッセージを生成します。

```
CREATE PROCEDURE DBA.StockLevel( @product_ID INTEGER ) AS
BEGIN
  DECLARe @qty INTEGER
  SELECT Quantity INTO @qty FROM 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;
```

CHECKPOINT 文

データベースにチェックポイントを設定します。

構文

CHECKPOINT

備考

CHECKPOINT 文はデータベースサーバーにチェックポイントの実行を強制します。またチェックポイントは、内部アルゴリズムに従ってデータベースサーバーによっても自動的に実行されます。通常、アプリケーションが、CHECKPOINT 文を発行する必要はありません。

パーミッション

ネットワークデータベースサーバーに CHECKPOINT 文を実行するには、DBA 権限が必要です。

パーソナルデータベースサーバーに CHECKPOINT 文を実行するには、パーミッションは必要ありません。

関連する動作

なし

参照

- 「バックアップとデータリカバリ」『SQL Anywhere サーバー データベース管理』
- 「チェックポイントログ」『SQL Anywhere サーバー データベース管理』
- 「checkpoint_time オプション」『SQL Anywhere サーバー データベース管理』
- 「recovery_time オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　CHECKPOINT 文は Adaptive Server Enterprise によってサポートされます。

CLEAR 文 [Interactive SQL]

Interactive SQL で開いているすべての結果セットを閉じます。

構文

CLEAR

備考

開いているすべての結果セットを閉じて、[SQL 文] ウィンドウ枠の内容はそのままにします。

パーミッション

なし

関連する動作

クリアされているデータに関連付けられているカーソルを閉じます。

参照

- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

CLOSE 文 [ESQL] [SP]

カーソルを閉じます。

構文

CLOSE *cursor-name*

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

cursor-name : *identifier* | *hostvar*

備考

この文は指定したカーソルを閉じます。

パーミッション

事前にカーソルを開いておきます。

関連する動作

なし

参照

- 「OPEN 文 [ESQL] [SP]」844 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「PREPARE 文 [ESQL]」856 ページ

標準と互換性

- **SQL/2008**　コア機能。Embedded SQL で使用される CLOSE 文は、オプション言語機能 B031 (Basic dynamic SQL) の一部です。
- **Transact-SQL**　Adaptive Server Enterprise によってサポートされます。

例

次の例は Embedded SQL のカーソルを閉じます。

```
EXEC SQL CLOSE employee_cursor;
EXEC SQL CLOSE :cursor_var;
```

次のプロシージャーはカーソルを使用します。

```
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 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
```

COMMENT 文

データベースオブジェクトに対するコメントをシステムテーブルに格納します。

構文

COMMENT ON {
  COLUMN [*owner*.]*table-name.column-name*
 | **DBSPACE** *dbspace-name*
 | **EVENT** [*owner*.]*event-name*
 | **EXTERNAL ENVIRONMENT** *environment-name*
 | **EXTERNAL** [**ENVIRONMENT**] **OBJECT** *object-name*
 | **FOREIGN KEY** [*owner*.]*table-name.role-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***"**
 | **LOGIN POLICY** *policy-name*
 | **MATERIALIZED VIEW** [*owner*.]*materialized-view-name*
 | **MIRROR SERVER** *mirror-server-name*
 | **PRIMARY KEY ON** [*owner*.]*table-name*
 | **PROCEDURE** [*owner*.]*procedure-name*
 | **SEQUENCE** *sequence-name*
 | **SERVICE** *web-service-name*
 | **SPATIAL REFERENCE SYSTEM** *srs-name*
 | **SPATIAL UNIT OF MEASURE** *uom-identifier*
 | **TABLE** [*owner*.]*table-name*
 | **TEXT CONFIGURATION** [*owner*.]*text-config-name*
 | **TEXT INDEX** *text-index-name* **ON** [*owner*.]*table-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**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

COMMENT 文は、データベース内のオブジェクトに注釈 (コメント) を設定するときに使用します。COMMENT 文を使用すると、ISYSREMARKS システムテーブル内の注釈が更新されます。コメントは、NULL に設定すると削除できます。インデックスまたはトリガーに対するコメントの場合、コメントの所有者はインデックスまたはトリガーが定義されているテーブルの所有者です。

ローカルのテンポラリテーブルにはコメントを追加できません。

データベースドキュメントウィザードを使用して SQL Anywhere データベースをドキュメント化する場合、プロシージャー、関数、トリガー、イベント、ビューのコメントを出力に含めるオプションがあります。

パーミッション

コメントされるデータベースオブジェクトの作成者であるか、または DBA 権限が必要です。

関連する動作

オートコミット。

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　Adaptive Server Enterprise によってサポートされません。

例

次の例は、コメントの追加と削除の方法を示します。

1. Employees テーブルにコメントを追加します。

```
COMMENT
ON TABLE Employees
IS 'Employee information';
```

2. Employees テーブルからコメントを削除します。

```
COMMENT
ON TABLE Employees
IS NULL;
```

オブジェクトのコメント設定を表示するには、次のような SELECT 文を使用します。この例は、SQL Anywhere サンプルデータベースの ViewSalesOrders ビューについてコメント設定を取得します。

```
SELECT remarks
FROM SYSTAB t, SYSREMARK r
WHERE t.object_id = r.object_id
AND t.table_name = 'ViewSalesOrders';
```

参照

- 「データベースのドキュメント化」『SQL Anywhere サーバー データベース管理』

COMMIT 文

データベースを永続的に変更したり、ユーザー定義のトランザクションを終了したりします。

構文 1

COMMIT [**WORK**]

構文 2

COMMIT TRAN[SACTION] [*transaction-name*]

パラメーター

transaction-name このトランザクションに割り当てられた名前です (任意)。有効な識別子を指定します。ネストされている BEGIN/COMMIT または BEGIN/ROLLBACK 文の最も外側のペアだけでトランザクション名を使用してください。

Adaptive Server Enterprise と SQL Anywhere のトランザクションネストの詳細については、「BEGIN TRANSACTION 文 [T-SQL]」500 ページを参照してください。

さまざまなオプションを使用して、COMMIT 文の動作を詳細に制御できます。次の項を参照してください。

- 「cooperative_commit_timeout オプション」『SQL Anywhere サーバー データベース管理』
- 「cooperative_commits オプション」『SQL Anywhere サーバー データベース管理』
- 「delayed_commits オプション」『SQL Anywhere サーバー データベース管理』
- 「delayed_commit_timeout オプション」『SQL Anywhere サーバー データベース管理』
- 「永続的なデータ変更」『SQL Anywhere サーバー SQL の使用法』

現在の接続のコミット数を返すには、Commit 接続プロパティを使用できます。

備考

- **構文 1** COMMIT 文は、トランザクションを終了し、このトランザクション中で行われたすべての変更内容をデータベースに永続化します。

  すべてのデータ定義文で、コミットが自動的に実行されます。詳細については、各 SQL 文の「関連する動作」を参照してください。

  データベースサーバーが無効な外部キーを検知すると、COMMIT 文は失敗します。この動作によって、無効な外部キーを持つトランザクションは終了できなくなります。通常、外部キー整合性をそれぞれのデータ操作オペレーションごとにチェックします。ただし、データベースオプション wait_for_commit が On に設定されているか、または特定の外部キーが CHECK ON COMMIT 句によって定義されている場合、データベースサーバーは COMMIT オプションが実行されるまで整合性のチェックを遅延します。

  COMMIT 単独での使用は、COMMIT WORK と同じです。

- **構文 2** BEGIN TRANSACTION 文と COMMIT TRANSACTION 文をペアで使用すると、ネストされたトランザクションを作成できます。ネストされたトランザクションはセーブポイントに似ています。COMMIT 文がネストされたトランザクションの一番外側で実行された場

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

合は、データベースに対する変更が保存されます。トランザクション内で実行する場合、COMMIT TRANSACTION はトランザクションのネストされているレベルを 1 つずつ減らします。トランザクションがネストされているときは、最も外側の COMMIT だけがデータベースへの変更を保存します。

構文 2 は Transact-SQL 拡張です。

Interactive SQL では、次の方法でも COMMIT を実行できます。

●[Ctrl + Shift + C] キーを押す。

●**[SQL]** » **[コミット]** をクリックします。

Interactive SQL で、**[SQL]** メニューから、またはキーボードショートカットで COMMIT を実行しても、**[SQL 文]** ウィンドウ枠の内容は変更されませんが、**[結果]** ウィンドウ枠の **[結果]** タブはクリアされます。

パーミッション

なし

関連する動作

WITH HOLD によって開かれたカーソルを除き、すべてのカーソルを閉じます。

この接続で宣言されたテンポラリテーブルのすべてのローを削除します。ただし、宣言に ON COMMIT PRESERVE ROWS が指定されているテーブルは除きます。

データベースでトランザクションログを使用していない場合、COMMIT 操作ごとに暗黙的なチェックポイントが発生します。

参照

- 「wait_for_commit オプション」『SQL Anywhere サーバー データベース管理』
- 「auto_commit オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「commit_on_exit オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL での COMMIT 文と ROLLBACK 文の実行」『SQL Anywhere サーバー データベース管理』
- 「SAVEPOINT 文」895 ページ
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「BEGIN TRANSACTION 文 [T-SQL]」500 ページ
- 「PREPARE TO COMMIT 文」858 ページ
- 「ROLLBACK 文」890 ページ

標準と互換性

- **SQL/2008**　構文 1 はコア機能です。構文 2 は Transact-SQL 拡張です。

例

次の文は現在のトランザクションをコミットします。

```
COMMIT;
```

次の Transact-SQL バッチは、@@trancount の値を、連続した値 0、1、2、1、0 で報告します。

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
go
```

CONFIGURE 文 [Interactive SQL]

Interactive SQL の **[オプション]** ウィンドウを開きます。

構文

CONFIGURE

備考

CONFIGURE 文は、Interactive SQL の **[オプション]** ウィンドウを開きます。このウィンドウには、すべての Interactive SQL オプションの現在の設定が表示されます。データベースオプションへの変更は、表示も許可もしません。このウィンドウで Interactive SQL を設定できます。

パーミッション

なし

関連する動作

なし

参照

- 「Interactive SQL のカスタマイズ」『SQL Anywhere サーバー データベース管理』
- 「SET OPTION 文」913 ページ
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

CONNECT 文 [ESQL] [Interactive SQL]

データベースへの接続を確立します。

構文 1 - 共有メモリ接続

CONNECT
[**TO** *database-server-name*]

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

[**DATABASE** *database-name*]
[**AS** *connection-name*]
[**USER**] *userid* [**IDENTIFIED BY** *password*]

database-server-name, *database-name*, *connection-name*, *userid*, *password* :
{ *identifier* | *string* | *hostvar* }

構文 2 - TCP/IP 接続

CONNECT USING *connect-string*

connect-string : { *identifier* | *string* | *hostvar* }

パラメーター

AS 句　オプションとして AS 句を指定して、接続に名前を付けることができます。名前を付けると、同じデータベースへの複数の接続、あるいは同じまたは異なるデータベースサーバーへの複数の接続が、すべて同時に行えるようになります。それぞれの接続には、固有のトランザクションがあります。たとえば、2 つの異なる接続から同じデータベース内の同じレコードを修正しようとする場合は、トランザクション間でロックの競合が起こることもあります。

構文 2 の場合、*connect-string* はセミコロンで区切った keyword=value 形式のパラメーター設定リストです。一重引用符で囲んでください。

備考

CONNECT 文は、*database-server-name* によって識別されるデータベースサーバー上で稼働している、*database-name* によって識別されるデータベースへの接続を確立します。この文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

構文 1 は、同じコンピューター上で実行しているデータベースサーバーへの共有メモリ接続に対してだけサポートされます。TCP/IP を使用してローカルデータベースサーバーに接続するか、または別のコンピューター上で実行しているデータベースサーバーに接続する場合は、構文 2 を使用してください。

- **Embedded SQL の動作**　Embedded SQL では、*database-server-name* を指定しない場合、デフォルトのローカルデータベースサーバー (最初に起動するデータベースサーバー) が使用されます。*database-name* を指定しない場合、指定されたサーバー上の最初のデータベースが使用されます。

  WHENEVER 文、SET SQLCA 文、一部の DECLARE 文はコードを生成しないので、ソースファイル内の CONNECT 文の前に置いてもかまいません。それ以外の場合は、CONNECT 文が正しく実行されるまで、どのような文も使用できません。

  ユーザー ID とパスワードを使って、動的 SQL 文ごとにパーミッションをチェックします。

  > **注意**
  > SQL Anywhere の場合、Embedded SQL では構文 1 のみ有効です。Ultra Light の場合、Embedded SQL では構文 1 と構文 2 の両方を使用できます。

- **Interactive SQL の動作**　CONNECT 文でデータベースまたはサーバーを指定しない場合、Interactive SQL はデフォルトのサーバーとデータベースには接続しないで、現在のデータ

ベースとの接続を継続します。サーバー名を指定せずに、データベース名を指定する場合、Interactive SQL は現在のサーバーの指定したデータベースに接続しようとします。データベース名を指定せずにサーバー名を指定する場合、Interactive SQL は指定したサーバーのデフォルトデータベースに接続します。

たとえば、データベースに接続した状態で次のようなバッチを実行すると、同じデータベースに 2 つのテーブルが作成されます。

```
CREATE TABLE t1( c1 int );
CONNECT DBA IDENTIFIED BY sql;
CREATE TABLE t2 (c1 int );
```

CONNECT 文が正しく実行されるまで、他のデータベース文を使用できません。

Interactive SQL をウィンドウモードで実行している場合、接続パラメーターが足りないというプロンプトが表示されます。

Interactive SQL がコマンドプロンプトモード (-nogui はコマンドラインから Interactive SQL を開始したときに指定します) またはバッチモードで実行中の場合、または AS 句を付けずに CONNECT を実行した場合は、無名の接続が開かれます。別の無名の接続がすでに開いている場合には、古い接続は自動的に閉じられます。それ以外の場合は、CONNECT 文を実行しても、既存の接続は閉じられません。

複数の接続も、現在の接続という概念を使って管理されます。接続文が成功した後、新しい接続が現在の接続になります。別の接続に切り替えるには、SET CONNECTION 文を使います。DISCONNECT 文を使って接続を切断することができます。

Interactive SQL に接続するとき、CONNECT [USER] *userid* を指定することは、SETUSER WITH OPTION *userid* 文を実行することと同じです。

Interactive SQL では、接続情報 (データベース名、ユーザー ID、データベースサーバーなど) は、[SQL 文] ウィンドウ枠の上のタイトルバーに表示されます。データベースに接続していない場合は、タイトルバーに [未接続] と表示されます。

注意
Interactive SQL では構文 1 と構文 2 の両方が有効です。ただし、Interactive SQL では *hostvar* 引数をサポートしていないことに注意してください。

パーミッション

なし

関連する動作

なし

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「GRANT 文」782 ページ
- 「DISCONNECT 文 [ESQL] [Interactive SQL]」707 ページ
- 「SET CONNECTION statement [Interactive SQL] [ESQL]」906 ページ
- 「SETUSER 文」932 ページ
- 「接続のトラブルシューティング」『SQL Anywhere サーバー データベース管理』
- 「接続パラメーター」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　構文 1 は、SQL/2008 標準のオプション言語機能 F771 です。構文 2 はベンダー拡張です。

- **Transact-SQL**　構文 1 と構文 2 はどちらも Adaptive Server Enterprise によってサポートされます。

例

次は、Embedded SQL 内での CONNECT 使用の例です。

```
EXEC SQL CONNECT AS :conn_name
USER :userid IDENTIFIED BY :password;
EXEC SQL CONNECT USER "DBA" IDENTIFIED BY "sql";
```

次の例では、SQL Anywhere サンプルデータベースが起動していることを前提としています。

Interactive SQL からデータベースに接続します。Interactive SQL がユーザー ID とパスワードを要求するプロンプトを表示します。

```
CONNECT;
```

DBA として Interactive SQL からデフォルトデータベースに接続します。Interactive SQL はパスワードを要求するプロンプトを表示します。

```
CONNECT USER "DBA";
```

DBA として Interactive SQL からサンプルデータベースに接続します。

```
CONNECT
TO demo12
USER DBA
IDENTIFIED BY sql;
```

接続文字列を使って Interactive SQL からサンプルデータベースに接続します。

```
CONNECT
USING 'UID=DBA;PWD=sql;DBN=demo';
```

サンプルデータベースに接続すると、タイトルバーにデータベース名、ユーザー ID、データベースサーバー名が **demo (DBA) on demo12** と表示されます。

CONTINUE 文

ループを再開します。

構文

CONTINUE [*statement-label*]

備考

CONTINUE 文は、ループを再開するための制御文です。実行はループの後に記述されている最初の文から継続されます。一連の Watcom-SQL 文の中で CONTINUE を使用する場合は、*statement-label* を指定してください。

Transact-SQL を使用する一連の文の中で CONTINUE を使用する場合は、*statement-label* を使用しないでください。

パーミッション

なし

関連する動作

なし

参照

- 「LOOP 文」832 ページ
- 「WHILE 文 [T-SQL]」988 ページ
- 「FOR 文」753 ページ
- 「BEGIN 文」497 ページ
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　文ラベルを使用しない CONTINUE は、Adaptive Server Enterprise によってサポートされます。

例

次のフラグメントは、CONTINUE 文を使ってループを再開する方法を示します。この例は、1 ～ 10 の間の奇数を表示します。

```
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;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
  END LOOP lbl;
END
```

CREATE DATABASE 文

データベースを作成します。

構文

CREATE DATABASE *db-filename-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** }]
[**DBA USER** *userid*]
[**DBA PASSWORD** *password*]
[**ENCODING** *encoding-label*]
[**ENCRYPTED** [**TABLE**] { *algorithm-key-spec* | **OFF** }]
[**JCONNECT** { **ON** | **OFF** }]
[**PAGE SIZE** *page-size*]
[**NCHAR COLLATION** *nchar-collation-label*[**(** *collation-tailoring-string* **)**]]
[[**TRANSACTION**] { **LOG OFF** | **LOG ON** [*log-filename-string*]
[**MIRROR** *mirror-filename-string*] }]

page-size :
2048 | **4096** | **8192** | **16384** | **32768**

algorithm-key-spec :
ON
| [**ON**] **KEY** *key* [**ALGORITHM** *AES-algorithm*]
| [**ON**] **ALGORITHM** *AES-algorithm* **KEY** *key*
| [**ON**] **ALGORITHM 'SIMPLE'**

AES-algorithm :
'AES' | **'AES256'** | **'AES_FIPS'** | **'AES256_FIPS'**

key : 引用符付き文字列

パラメーター

CREATE DATABASE　ファイル名 (*db-filename-string*、*log-filename-string*、*mirror-filename-string*) はオペレーティングシステムファイル名を含む文字列です。リテラル文字列として、一重引用符で囲んでください。

○パスを指定する場合、後ろに n または x が続くすべての円記号 (¥) は、2 つ重ねます。このようにエスケープすることによって、SQL の文字列のルールに従って、改行文字 (¥n) または 16 進数字 (¥x) として解釈されるのを回避できます。

常に円記号をエスケープすると安全です。次に例を示します。

```
CREATE DATABASE 'c:¥¥databases¥¥my_db.db'
LOG ON 'e:¥¥logdrive¥¥my_db.log';
```

○パスを指定しない場合、または相対パスで指定する場合、データベースファイルはデータベースサーバーの作業ディレクトリを基準に作成されます。トランザクションログファイルのパスを指定しないと、データベースファイルと同じディレクトリにファイルが作成されます。データベースファイルとトランザクションログを、コンピューターの別々のディスクに保存することをおすすめします。

○ファイル拡張子の指定がない場合、作成されるデータベースファイルには *.db* 拡張子、トランザクションログには *.log* 拡張子、トランザクションログミラーには *.mlg* 拡張子が付けられます。

db-filename-string には utility_db を指定できません。この名前は、ユーティリティデータベースのために予約されています。

ACCENT 句　この句は、データベースでのアクセント記号の区別を指定するときに使用します。この句のサポートは廃止されました。アクセント記号の区別を指定するには、COLLATION 句と NCHAR COLLATION 句に提供されている照合の適合化オプションを使用してください。

ACCENT 句は、COLLATION 句または NCHAR COLLATION 句で指定した照合に UCA (Unicode 照合アルゴリズム) を使用する場合にのみ適用されます。ACCENT RESPECT 句を使用すると、文字のアクセント記号を考慮して UCA 文字列が比較されます。たとえば、e は é よりも小さいと扱われます。ACCENT FRENCH は ACCENT RESPECT と似ていますが、フランス語の規則に合わせてアクセント記号が右から左の方向で比較される点が異なります。ACCENT IGNORE 句を使用すると、アクセント記号を無視して文字列が比較されます。たとえば、e と é は同等です。

データベースの作成時にアクセント記号の区別が指定されていない場合、比較とソートでのアクセント記号の区別は、デフォルトでは**区別しない**になります。ただし、UCA 照合を使用して作成された日本語のデータベースは例外であり、デフォルトのアクセント記号の区別は**区別する**になります。

ASE COMPATIBLE 句　SYS.SYSCOLUMNS ビューと SYS.SYSINDEXES ビューを作成しません。デフォルトでは、これらのビューは Watcom SQL で使用可能なシステムテーブルとの互換性を保つために作成されます (このソフトウェアのバージョン 4 以前)。これらのビューは、Adaptive Server Enterprise の互換ビュー dbo.syscolumns および dbo.sysindexes と競合します。

BLANK PADDING 句　SQL Anywhere は、文字列について、可変長であり VARCHAR ドメインを使用して格納されている文字列と同じ扱いで、すべての文字列を比較します。これには、固定長の CHAR カラムまたは NCHAR カラムの文字列比較も含まれます。また、値がデータベースに格納されている場合、SQL Anywhere は後続ブランクのトリムや埋め込みは行いません。

デフォルトでは、SQL Anywhere はブランクを意味のある文字として扱います。たとえば、値 'a ' (文字 'a' と、後続の 1 つのブランク) は、単一文字の文字列 'a' とは等しくありません。不等号比較の照合でも、ブランクは、他の文字と同じように扱われます。

ブランク埋め込みが有効である場合 (BLANK PADDING ON を指定)、文字列比較のセマンティックは ANSI/ISO SQL 標準とさらに近づきます。ブランク埋め込みが有効であると、SQL Anywhere はどのような比較であっても後続ブランクを無視します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

上に挙げた例では、ブランクを埋め込まれたデータベースで 'a ' を 'a' に対して等号比較すると、TRUE が返されます。ブランクを埋め込まれたデータベースでは、固定長文字列の値は、アプリケーションによってフェッチされたときにブランクで埋め込まれます。このような文字の割り当てが行われたときにアプリケーションが文字列のトランケーション警告を受け取るかどうかは、ansi_blanks 接続オプションによって制御されます。

CASE 句　この句は、データベースでの大文字と小文字の区別を指定するときに使用します。この句のサポートは廃止されました。大文字と小文字の区別を指定するには、COLLATION 句と NCHAR COLLATION 句に提供されている照合の適合化オプションを使用してください。

CASE RESPECT 文を使用すると、すべての CHAR データ型と NCHAR データ型を比較するときに大文字と小文字が区別されます。UCA を使用した比較では、元の文字とアクセント記号がすべて同じ場合にのみ、大文字と小文字の違いが考慮されます。その他の照合の場合、大文字と小文字が区別されます。たとえば、a は A よりも小さく、A は b よりも小さいというように扱われます。CASE IGNORE 句を使用すると、大文字と小文字を区別せずに文字列が比較されます。大文字と小文字は同一と見なされます。

データベースの作成時に大文字と小文字の区別が指定されていない場合、比較とソートでの大文字と小文字の区別は、デフォルトでは**区別しない**になります。ただし、UCA 照合を使用して作成された日本語のデータベースは例外であり、デフォルトの大文字と小文字の区別は**区別する**になります。

CASE RESPECT は、ISO/ANSI SQL 標準との互換性を保つために用意されています。大文字と小文字を区別するデータベースであっても、データベースの識別子については大文字と小文字は常に区別されません。

CHECKSUM 句　チェックサムは、データベースページがディスク上で変更されたかどうかを判断するために使用します。グローバルチェックサムを有効にしてデータベースを作成した場合、チェックサムはページがディスクに書き込まれる直前に計算されます。そのページが次にディスクから読み出されるときに、ページのチェックサムが再計算されて、ページに保存されているチェックサムと比較されます。チェックサムが異なる場合は、ディスク上でページが変更されており、エラーが発生します。グローバルチェックサムを有効にして作成されたデータベースも、チェックサムを使用して検証されます。次の文を実行することによって、データベースがグローバルチェックサムを有効にして作成されたかどうかをチェックできます。

```
SELECT DB_PROPERTY ( 'Checksum' );
```

グローバルチェックサムがオンの場合、このクエリは ON を返します。それ以外の場合は OFF を返します。デフォルトでグローバルチェックサムはオンです。そのため、CHECKSUM 句を省略すると ON が適用されます。

リムーバブルドライブなどのストレージデバイス上で実行されているデータベースや Windows Mobile 上で実行されているデータベースでは、データベースファイルの破損を速やかに検出できるように、この句の設定に関係なく、データベースサーバーによって常に書き込みチェックサムが有効になります。また、検証アクティビティの実行時に重要なページのチェックサムも計算されます。

グローバルチェックサムが有効になっていないデータベースでは、-wc オプションを使用して書き込みチェックサムを有効にすることができます。

COLLATION 句　COLLATION で指定した照合は、文字データ型 (CHAR、VARCHAR、LONG VARCHAR) のソートと比較に使用されます。照合は、使用されるエンコード (文字セット) に文字の比較と順序付けに関する情報をもたらすものです。COLLATION 句が指定されていない場合、SQL Anywhere はオペレーティングシステムの言語とエンコードに基づいて照合を選択します。

照合は、SQL Anywhere Collation Algorithm (SACA) を使用する照合リストから選択するか、Unicode 照合アルゴリズム (UCA) にすることができます。UCA を指定する場合、ENCODING 句も指定してください。

照合は慎重に選択してください。データベースの作成後に照合は変更できません。

オプションで、文字列のソートや比較を詳細に制御することを目的に、照合の適合化オプション (*collation-tailoring-string*) を指定できます。これらのオプションは、キーワード=値 の形式で、カッコで囲んで指定して、その後ろに照合名を記述します。たとえば、... CHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)' のように記述します。

DATABASE SIZE 句　このオプションの句を使用して、データベースファイルの初期サイズを設定します。単位をキロバイト、メガバイト、ギガバイト、またはページで指定するには、それぞれ KB、MB、GB、または PAGES を使用します。

作成時におけるファイルサイズの指定は、ファイル用領域の事前割り付けを行う方法です。このようにすると、データベースがあるドライブの空き領域が不足する危険性を小さくすることができます。また、データベースサイズを拡大する操作は時間を要するため、それが必要となる前にデータベースに保存できるデータの量を増やすことがパフォーマンスの向上につながります。

DBA USER 句　この句は、データベースの DBA ユーザーを指定するときに使用します。この句を使用すると、デフォルトの DBA ユーザー権限ではデータベースに接続できなくなります。この句を指定しない場合、デフォルトの DBA ユーザー ID が作成されます。

DBA PASSWORD 句　DBA データベースユーザーに別のパスワードを指定することができます。この句を指定しない場合は、デフォルトのパスワード (**sql**) が DBA ユーザーに使用されます。

○ユーザー ID には、次に該当する値を指定できません。

- 空白スペース、一重引用符、または二重引用符で始まる値
- 空白スペースで終わる値
- セミコロンを含む値

○パスワードは大文字と小文字が区別され、次に該当する値は指定できません。

- 空白スペース、一重引用符、または二重引用符で始まる値
- 空白スペースで終わる値
- セミコロンを含む値
- 256 バイト長を超える値

ENCODING 句　COLLATION 句で指定するほとんどの照合には、エンコード (文字セット) と順序付けの両方が定義されています。そのような照合については、ENCODING 句を指定する必要はありません。ただし、COLLATION 句に指定されている値が UCA (Unicode 照合アルゴリズム)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

の場合、ENCODING を使用してロケール固有のエンコードを指定し、比較と順序指定に UCA を活用できます。ENCODING 句では、CHAR データ型に UTF-8 または任意のシングルバイトエンコードを指定できます。ENCODING は、UTF-8 以外のマルチバイトエンコードを指定できません。

UCA 照合を選択した場合は、オプションで照合の適合化オプションを指定できます。

COLLATION が UCA に設定され、ENCODING が指定されていない場合、SQL Anywhere では UTF-8 が使用されます。

ENCRYPTED 句または ENCRYPTED TABLE 句　暗号化すると、格納データを読み取ることができなくなります。データベース全体を暗号化するには、ENCRYPTED キーワード (TABLE なし) を使用します。テーブルの暗号化を有効にするときは、ENCRYPTED TABLE 句を使用します。「テーブルの暗号化を有効にする」とは、以降にキーワード ENCRYPTED 句を使用して作成されるテーブルや変更されるテーブルは、データベースの作成時に指定した設定を使用して暗号化されるということです。

データベースとテーブルの暗号化には単純と強力という 2 つのレベルがあります。単純暗号化は、難読化と同じです。データは判読できませんが、暗号に関する知識を持ったユーザーはデータを解読できます。強力な暗号化を使用すると、データは判読不能になり、事実上は解読できません。

単純暗号化の場合は、ENCRYPTED ON ALGORITHM SIMPLE または ENCRYPTED ALGORITHM SIMPLE を指定するか、アルゴリズムやキーを指定せずに ENCRYPTED ON 句を指定します。

強力な暗号化では、128 ビットまたは 256 ビットの AES アルゴリズムを使用する ENCRYPTED ON ALGORITHM と KEY 句を指定して暗号化キーを指定します。キーには最低でも 16 文字の値を選択し、大文字と小文字、数字、文字、特殊文字を組み合わせて使用することをおすすめします。

Windows Mobile では、ARM プロセッサー用に AES_FIPS および AES256_FIPS アルゴリズムのみがサポートされています。

警告
強力な暗号化が適用されたデータベースの場合、キーのコピーは必ず安全な場所に保管してください。暗号化キーがわからなくなった場合は、Sybase 製品の保守契約を結んでいるサポートセンタに依頼してもデータにはアクセスできません。アクセスできなくなったデータベースは、廃棄して、新しくデータベースを作成する必要があります。

既存のデータベースの暗号化されたコピーは、CREATE ENCRYPTED DATABASE 文を使用して作成することもできます。

JCONNECT 句　jConnect JDBC ドライバーからシステムカタログ情報にアクセスできるようにするには、JCONNECT ON を指定します。この句によって jConnect をサポートするシステムオブジェクトがインストールされます。jConnect システムオブジェクトを含まない場合、JCONNECT OFF を指定します。その場合でも、システム情報にアクセスしないかぎり、JDBC を使用できます。JCONNECT はデフォルトで ON です。

PAGE SIZE 句　データベースのページサイズは、2048、4096、8192、16384、または 32768 バイトです。デフォルトのページサイズは 4096 バイトです。ページサイズを大きくすると、一般に大規模なデータベースのパフォーマンスは向上しますが、オーバーヘッドは増大します。

次に例を示します。

```
CREATE DATABASE 'c:¥¥databases¥¥my_db.db'
PAGE SIZE 4096;
```

> **注意**
> 現在のサーバーで使用しているページサイズより大きいページサイズは指定できません。サーバーページサイズは、最初に起動されたデータベースのセットから取得されるか、サーバーコマンドラインで -gp オプションを使用して設定します。

NCHAR COLLATION 句　NCHAR COLLATION 句で指定した照合は、各国の文字データ型 (NCHAR、NVARCHAR、LONG NVARCHAR) のソートと比較に使用されます。照合は、各国の文字に使用される UTF-8 エンコード (文字セット) に文字の順序付けに関する情報をもたらすものです。NCHAR COLLATION 句が指定されない場合、SQL Anywhere は Unicode 照合アルゴリズム (UCA) を使用します。その他に使用できる照合は UTF8BIN のみです。UTF8BIN は、エンコードが 0x7E を超えるすべての文字のバイナリ順を規定します。

オプションで、文字列のソートや比較を詳細に制御することを目的に、照合の適合化オプション (*collation-tailoring-string*) を指定できます。これらのオプションは、キーワード=値 の形式で、引用符で囲んで指定して、その後ろに照合名を記述します。たとえば、... NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)' のように記述します。ACCENT 句または CASE 句と、大文字小文字やアクセント記号の区別の設定を含む照合の適合化文字列とを、併せて指定すると、ACCENT 句と CASE 句の値はデフォルトとしてのみ使用されます。

> **注意**
> UCA 照合を指定すると、照合の適合化のすべてのオプションがサポートされます。その他の照合の場合、大文字小文字の区別の適合化オプションのみがサポートされます。
>
> 照合の適合化オプションを使用して作成したデータベースは、10.0.1 より前のデータベースサーバーでは起動できません。

TRANSACTION LOG 句　トランザクションログは、データベースサーバーがデータベースに対するすべての変更を記録するファイルです。トランザクションログはバックアップとリカバリ、データレプリケーションで重要な役割を果たします。

TRANSACTION 句の MIRROR 句を使用すると、トランザクションログのミラーを使用する場合にファイル名を指定できます。トランザクションログミラーはトランザクションログと同一のコピーで、通常は別のデバイスで管理され、データを確実に保護しています。デフォルトでは、SQL Anywhere はトランザクションログミラーを使用しません。

備考

指定された名前と属性でデータベースファイルを作成します。データベースはオペレーティングシステムファイルとして格納されます。この文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

この文を実行するのに必要なパーミッションは、サーバーコマンドラインで -gu オプションを使用して設定します。デフォルトの設定は、DBA 権限を必要とします。

データベースサーバーを実行中のアカウントには、ファイルが作成されたディレクトリの書き込みパーミッションが必要です。

関連する動作

オペレーティングシステムファイルが作成されます。

参照

- 「ALTER DATABASE 文」408 ページ
- 「DROP DATABASE 文」709 ページ
- 「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバー データベース管理』
- 「DatabaseKey (DBKEY) 接続パラメーター」『SQL Anywhere サーバー データベース管理』
- 「トランザクションログ」『SQL Anywhere サーバー データベース管理』
- 「検証ユーティリティ (dbvalid)」『SQL Anywhere サーバー データベース管理』
- 「チェックサムを使用した破損の検出」『SQL Anywhere サーバー データベース管理』
- 「VALIDATE 文」983 ページ
- 「CREATE ENCRYPTED DATABASE 文」535 ページ
- 「テーブル暗号化」『SQL Anywhere サーバー データベース管理』
- 「ユーティリティデータベース」『SQL Anywhere サーバー データベース管理』
- 「ansi_blanks オプション」『SQL Anywhere サーバー データベース管理』
- 「-gp dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「-wc dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「-wc dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「sa_validate システムプロシージャー」1210 ページ
- 「照合の適合化オプション」『SQL Anywhere サーバー データベース管理』
- 「照合の考慮事項」『SQL Anywhere サーバー データベース管理』
- 「サポートされている照合と代替照合」『SQL Anywhere サーバー データベース管理』
- 「強力な暗号化」『SQL Anywhere サーバー データベース管理』
- 「Windows Mobile の jConnect」『SQL Anywhere サーバー データベース管理』
- 「国際言語と文字セット」『SQL Anywhere サーバー データベース管理』
- 「推奨文字セットと照合」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

- **Transact-SQL**　CREATE DATABASE 文は Adaptive Server Enterprise でサポートされていますが、使用する句は異なります。

例

次の文は、*C:¥*ディレクトリにデータベースファイル *mydb.db* を作成します。

```
CREATE DATABASE 'C:¥¥mydb.db'
TRANSACTION LOG ON
CASE IGNORE
PAGE SIZE 4096
```

```
ENCRYPTED OFF
BLANK PADDING OFF;
```

次の文は、コードページ 1252 を使用してデータベースを作成し、CHAR と NCHAR のデータ型の両方に UCA を使用します。比較とソート時に、アクセント記号と大文字と小文字の区別が考慮されます。

```
CREATE DATABASE 'c:¥¥uca.db'
COLLATION 'UCA'
ENCODING 'CP1252'
NCHAR COLLATION 'UCA'
ACCENT RESPECT
CASE RESPECT;
```

次の文はデータベース *myencrypteddb.db* を作成します。また、単純暗号化によって暗号化されます。

```
CREATE DATABASE 'myencrypteddb.db'
ENCRYPTED ON;
```

次の文はデータベース *mystrongencryptdb.db* を作成します。また、キー gh67AB2 を使用して暗号化されます (強力な暗号化)。

```
CREATE DATABASE 'mystrongencryptdb.db'
ENCRYPTED ON KEY 'gh67AB2';
```

次の文はデータベース *mytableencryptdb.db* を作成します。また、単純暗号化を使用してテーブルの暗号化が有効にされます。ENCRYPTED の後にキーワード TABLE を挿入した場合、データベースの暗号化ではなくテーブルの暗号化を示すことに注意してください。

```
CREATE DATABASE 'mytableencryptdb.db'
ENCRYPTED TABLE ON;
```

次の文はデータベース *mystrongencrypttabledb.db* を作成します。また、キー gh67AB2 (強力な暗号化) と AES_FIPS 暗号化アルゴリズムを使用してテーブルの暗号化が有効にされます。

```
CREATE DATABASE 'mystrongencrypttabledb.db'
ENCRYPTED TABLE ON KEY 'gh67AB2'
ALGORITHM 'AES_FIPS';
```

次の文は、照合 1252LATIN1 を使用するデータベースファイル *mydb.db* を作成します。NCHAR 照合を UCA に設定し、ロケールセットを es に設定して、大文字と小文字の区別とアクセント記号の区別を有効にします。

```
CREATE DATABASE 'my2.db'
    COLLATION '1252LATIN1(case=respect)'
    NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)';
```

CREATE DBSPACE 文

新しいデータベース領域を定義し、関連するデータベースファイルを作成します。

構文

CREATE DBSPACE *dbspace-name* **AS** *filename*

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

dbspace-name DB 領域の名前を指定します。これは、*filename* を使用して指定する実際のデータベースファイル名ではありません。*dbspace-name* は、文やプロシージャーなどで参照可能な内部名です。system、temporary、temp、translog、translogmirror は事前定義の DB 領域用に予約されているため、これらの名前を DB 領域の名前として使用することはできません。

ピリオド (.) を含む値を指定すると、エラーが返されます。

filename データベースファイルの名前を指定します。ファイルへのパスを含めることもできます。パスが指定されていない場合は、メインデータベースファイルと同じロケーション (ディレクトリ) にデータベースファイルが作成されます。別のロケーションを指定する場合、パスはデータベースサーバーを基準にした相対パスになります。円記号 (¥) は、SQL 文字列のエスケープ文字であるため、2 つ重ねます。

filename パラメーターには、文字列リテラルまたは変数のどちらかを指定してください。

「クラウドに関する注意点」クラウドのテナントデータベースについては、DB 領域の位置を指定する場合には、ファイル名だけを指定できます。ディレクトリパスを指定することはできません。

備考

CREATE DBSPACE 文は、新しいデータベースファイルを作成します。データベースの作成時には、ファイルは 1 つしかありません。作成されるすべてのテーブルとインデックスは、このファイルの中に入ります。CREATE DBSPACE は、新しいファイルをデータベースに追加します。このファイルはメインファイルではなく、別のディスクドライブ上にあることもあり、1 つの物理的なデバイスよりも大きなデータベースを作成できるようになります。

各データベースには、メインファイル以外に、最大で 12 の DB 領域という制限があります。

テーブル、インデックスなどの各オブジェクトは、1 つの DB 領域内にすべてが格納されます。CREATE 文の IN 句は、オブジェクトを入れる DB 領域を指定します。デフォルトでは、オブジェクトはシステムデータベースファイルの中に入ります。テーブルを作成する前に、default_dbspace オプションを設定することで、テーブルが作成される DB 領域を指定することもできます。

パーミッション

DBA 権限。

関連する動作

オートコミット。自動チェックポイント。

参照

- 「DROP DBSPACE 文」711 ページ
- 「default_dbspace オプション」『SQL Anywhere サーバー データベース管理』
- 「文字列」5 ページ
- 「事前定義の DB 領域」『SQL Anywhere サーバー データベース管理』
- 「追加の DB 領域の考慮事項」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の CREATE DBSPACE 文は、DB 領域 libbooks を作成します。DB 領域のデータベースファイル名は *library.db* であり、*c:*¥ディレクトリに格納されます。後続の CREATE TABLE 文は、libbooks DB 領域にテーブル LibraryBooks を作成します。

```
CREATE DBSPACE libbooks
AS 'c:¥¥library.db';
CREATE TABLE LibraryBooks (
  title char(100),
  author char(50),
  isbn char(30),
) IN libbooks;
```

CREATE DECRYPTED DATABASE 文

すべてのトランザクションログおよび DB 領域を含む既存のデータベースの、復号化されたコピーを作成します。

構文

CREATE DECRYPTED DATABASE *newfile*
FROM *oldfile*
[**KEY** *key*]

パラメーター

FROM 句　この句は、コピーするデータベースの名前 (*oldfile*) を指定するために使用します。

KEY 句　この句は、データベースの復号化に必要な暗号化キーを指定するために使用します。既存のデータベースの暗号化に使用されたのが、キーが不要な単純暗号化である場合は、KEY 句を指定しないでください。

備考

CREATE DECRYPTED DATABASE 文は、新しいデータベースファイル (*newfile*) を作成し、元のデータベースファイル (*oldfile*) の置換または削除はしません。

oldfile に含まれるすべての暗号化されたテーブルは、*newfile* では暗号化されず、テーブルの暗号化は有効になりません。

> **注意**
> SQL Anywhere 12 以降を使用して作成されたデータベースでは、不正なアクセスからデータを保護するため、ISYSCOLSTAT、ISYSUSER、ISYSEXTERNLOGIN の各システムテーブルは常に暗号化されたままになります。

oldfile でトランザクションログファイルまたはトランザクションログミラーが使用されている場合は、ファイル名がそれぞれ *newfile.log* および *newfile.mlg* に変更されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

oldfile に DB 領域ファイルが含まれている場合は、復号化を示す D がファイル名に付加されます。たとえば、CREATE DECRYPTED DATABASE 文を実行すると、*oldfile* が *mydbspace.dbs* なら、*newfile* は *mydbspace.dbsD* になります。

リカバリが必要なデータベースに対してこの文を実行することはできません。この文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

パーミッション

DBA 権限。

関連する動作

なし

参照

- 「データベースの暗号化と復号化」『SQL Anywhere サーバー データベース管理』
- 「CREATE ENCRYPTED DATABASE 文」535 ページ
- 「CREATE ENCRYPTED FILE 文」537 ページ
- 「CREATE DECRYPTED FILE 文」531 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の最初の文では、*demo.db* の AES256 で暗号化されたコピー *demoEncrypted.db* を作成します。2 番目の文では、*demoEncrypted.db* の復号化されたコピー *demoDecrypted.db* を作成します。

```
CREATE ENCRYPTED DATABASE 'demoEncrypted.db'
  FROM 'demo.db'
  KEY 'Sd8f6654*Mnn'
  ALGORITHM 'AES256';
CREATE DECRYPTED DATABASE 'demoDecrypted.db'
  FROM 'demoEncrypted.db'
  KEY 'Sd8f6654*Mnn';
```

CREATE DECRYPTED FILE 文

強力な暗号化が施されているデータベースの復号化されたコピーを作成します。この文を使用して、トランザクションログ、トランザクションログミラー、および DB 領域の復号化されたコピーを作成できます。

構文

CREATE DECRYPTED FILE *newfile*
FROM *oldfile* **KEY** *key*

パラメーター

FROM 句 暗号化ファイル名の一覧が表示されます。

KEY 句　暗号化されたファイルへのアクセスに必要なキーをリストします。

備考

この文は、データベースでリカバリが必要であり、データベースの復号化されたコピーをサポート目的で必要な場合に使用します。この文の使用は、トランザクションログファイル、トランザクションログミラーファイル、DB 領域などのデータベース関連ファイルの復号化でも必要です。

元のデータベースファイルは、暗号化キーを使用して強力な暗号化が施されている必要があります。復号化後のファイルは、暗号化されたファイルの正確なコピーですが、暗号化されていないため、暗号化キーは必要ありません。

この文を使用してデータベースを復号化する場合は、対応するトランザクションログファイル (と DB 領域) も復号化しなければ、データベースを使用できません。

リカバリを必要とするデータベースを復号化する場合は、対応するトランザクションログファイルの復号化および新しいデータベースのリカバリも必要です。このプロセスでトランザクションログファイルの名前が変わることはありません。したがって、データベースとトランザクションログファイルの名前を変更した場合は、復号化後のデータベースに対して dblog -t を実行する必要があります。

テーブルの暗号化が有効なデータベースでは、この文を使用できません。テーブルを復号化する場合、ALTER TABLE 文の NOT ENCRYPTED 句を使用して復号化します。

> **注意**
> SQL Anywhere 12 以降を使用して作成されたデータベースでは、データベースファイルへの不正なアクセスからデータを保護するため、ISYSCOLSTAT、ISYSUSER、ISYSEXTERNLOGIN の各システムテーブルは常に暗号化されたままになります。

この文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

パーミッション

DBA 権限。

関連する動作

なし

参照

- 「ALTER TABLE 文」462 ページ
- 「CREATE ENCRYPTED FILE 文」537 ページ
- 「CREATE DECRYPTED DATABASE 文」530 ページ
- 「CREATE ENCRYPTED DATABASE 文」535 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例は、contacts データベースを復号化し、暗号化されていない新しいデータベース contacts2 を作成します。

```
CREATE DECRYPTED FILE 'contacts2.db'
FROM 'contacts.db'
KEY 'Sd8f6654*Mnn';
```

CREATE DOMAIN 文

データベース内にドメインを作成します。

構文

CREATE { **DOMAIN** | **DATATYPE** } [**AS**] *domain-name data-type*
[[**NOT**] **NULL**]
[**DEFAULT** *default-value*]
[**CHECK** (*condition*)]

domain-name : *identifier*

data-type : *built-in data type*, *with precision and scale*

パラメーター

DOMAIN | DATATYPE 句　CREATE DOMAIN は SQL/2008 標準で定義されているため、CREATE DATATYPE ではなく、CREATE DOMAIN を使用することをおすすめします。

NULL 句　この句を使用すると、ドメインに NULL 入力可かどうかを指定できます。ドメインを使用してカラムを定義する場合、NULL 入力可かどうかは次のようにして決まります。

○カラム定義で指定された Null 入力可能。

○ドメイン定義で指定された Null 入力可能。

○NULL 入力属性がカラム定義とドメイン定義のどちらでも明示的に指定されていない場合、allow_nulls_by_default オプションの設定が使用されます。

CHECK 句　検査制約を使用してドメインを作成する場合は、検査制約の検索条件の中に @ 記号のプレフィクスを持つ変数名を使用できます。データ型をカラムの定義内で使う場合、このような変数をカラム名に置き換えます。これにより、そのドメインで定義された各テーブルにドメインの検査制約を適用することができます。

備考

ドメインは、必要に応じて精度と小数点以下の桁数を含めた組み込みデータ型のエイリアスです。データベース内の使いやすさを改善し、一貫性を高めます。

ドメインはデータベース内のオブジェクトです。名前を付けるには識別子のルールに従います。ドメインは、大文字と小文字を区別しません。組み込みデータ型名の場合と同じです。

データ型を作成するユーザーは、自動的にそのデータ型の所有者となります。CREATE DATATYPE 文の中では、所有者を指定できません。ドメイン名はユニークにします。また、すべてのユーザーはプレフィクスとして所有者を使わなくてもデータ型にアクセスできます。

ドメインは、CHECK 条件と DEFAULT 値を持つことができ、ユーザー側でそのデータ型が NULL 値を使えるかどうかを指定できます。これらの条件と値は、ドメインで定義するカラムによって継承されます。カラム定義で明示的に指定された条件または値は、ドメイン型に指定された条件または値を上書きします。

データベースからドメインを削除するには、DROP DOMAIN 文を使用します。ドメインを削除するには、ドメインの所有者であるか、DBA 権限が必要です。

パーミッション

RESOURCE 権限。

関連する動作

オートコミット。

参照

- 「allow_nulls_by_default オプション」『SQL Anywhere サーバー データベース管理』
- 「DROP DOMAIN 文」712 ページ
- 「SQL データ型」85 ページ

標準と互換性

- **SQL/2008** ドメインのサポートは、SQL/2008 標準のオプションの SQL 言語機能 F251 です。

例

次の文は、35 文字の文字列を保持し、NULL が使用できる address という名前のドメインを作成します。

```
CREATE DOMAIN address CHAR( 35 ) NULL;
```

次の文は、NULL は使用できず、デフォルトでオートインクリメントされる ID という名前のドメインを作成します。

```
CREATE DOMAIN ID INT
NOT NULL
DEFAULT AUTOINCREMENT;
```

次の文は、検査制約で正規表現を使用する PhoneNumber という名前のドメインを作成します。これは、文字列が、ダッシュまたは空白で区切られた 3 桁の市外局番、3 桁の局番、4 桁の電話番号で構成される、正しくフォーマットされた 12 文字の北米地域の電話番号であることを確認するためのものです。

```
CREATE DOMAIN PhoneNumber CHAR(12) NULL
CHECK( @PhoneNumber 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})');
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CREATE ENCRYPTED DATABASE 文

すべてのトランザクションログおよび DB 領域を含む既存のデータベースの、暗号化されたコピーを作成します。

構文 1 - データベースの暗号化されたコピーの作成

CREATE ENCRYPTED DATABASE *newfile*
FROM *oldfile*
[**KEY** *newkey*]
[**ALGORITHM** *algorithm*]
[**OLD KEY** *oldkey*]

algorithm :
'SIMPLE'
| **'AES'**
| **'AES256'**
| **'AES_FIPS'**
| **'AES256_FIPS'**

構文 2 - テーブル暗号化が有効にされたデータベースのコピーの作成

CREATE ENCRYPTED TABLE DATABASE *newfile*
FROM *oldfile*
[**KEY** *newkey*]
[**ALGORITHM** *algorithm*]
[**OLD KEY** *oldkey*]

パラメーター

CREATE ENCRYPTED DATABASE 句　この句は、暗号化される新しいデータベースの名前を指定するために使用します。

CREATE ENCRYPTED TABLE DATABASE 句　この句は、新しいデータベースの名前を指定するときに使用します。新しいデータベースは暗号化されませんが、テーブルの暗号化が有効になります。

FROM 句　この句は、元になるデータベースの名前 (*oldfile*) を指定するために使用します。

KEY 句　*algorithm* が SIMPLE 以外の場合は、この句を使用して、*newfile* の暗号化キーを指定します。

OLD KEY 句　この句は、*oldfile* の暗号化キーを指定するために使用します。この句が必要なのは、*oldfile* が単純暗号化 (SIMPLE) 以外で暗号化されている場合のみです。

ALGORITHM 句　この句は、*newfile* で使用する暗号化アルゴリズムを指定するために使用します。KEY 句を指定し、ALGORITHM 句を指定しない場合は、デフォルトで AES (128 ビット暗号化) が使用されます。*algorithm* に SIMPLE を指定する場合は、KEY 句を指定しません。

備考

また、この文を使用して、データベースのコピーを作成し、このコピーでテーブルの暗号化を有効にすることもできます。

oldfile には、暗号化されていないデータベース、暗号化されたデータベース、またはテーブル暗号化が有効なデータベースを指定できます。

構文 1 は、既存のデータベース *oldfile* から、その暗号化されたコピーである *newfile* を作成します。

構文 2 は、既存のデータベース *oldfile* から、そのコピー *newfile* を作成し、このコピーのテーブル暗号化を有効にします。この構文を使用する場合、*oldfile* で暗号化されていたすべてのテーブルは、*newfile* でも暗号化されます。*oldfile* に暗号化されているテーブルがない場合にテーブルを暗号化するには、暗号化の対象となるテーブルごとに ALTER TABLE...ENCRYPTED 文を実行できます。

どちらの構文を使用しても、*oldfile* が置換されたり削除されたりすることはありません。

oldfile でトランザクションログファイルまたはトランザクションログミラーファイルが使用されている場合は、名前がそれぞれ *newfile.log* および *newfile.mlg* に変更されます。

oldfile に DB 領域ファイルが含まれている場合は、暗号化を示す E がファイル名に付加されます。たとえば、CREATE ENCRYPTED DATABASE 文を実行すると、ファイル名 *mydbspace.dbs* は *mydbspace.dbsE* に変更されます。

この文を使用して、データベースの暗号化アルゴリズムとキーを変更できます。ただし、CREATE ENCRYPTED DATABASE 文は、新しいファイル (*newfile*) を作成しますが、前バージョンのファイル (*oldfile*) を置換または削除することはありません。

CREATE ENCRYPTED DATABASE 文と CREATE ENCRYPTED TABLE DATABASE 文は、リカバリを必要とするデータベースには実行できません。これらの文は、プロシージャー、トリガー、イベント、バッチではサポートされていません。

dbunload -an オプションに -ek または -ep を使用してデータベースのアンロードと再ロードを行うと、既存のデータベースを暗号化したり、既存の暗号化キーを変更したりできます。

暗号化されたデータベースや、テーブル暗号化が有効にされたデータベースは、CREATE DATABASE 文を使用して作成することもできます。

注意
FIPS 認定の暗号化をサポートしないプラットフォームもあります。サポートされるプラットフォームのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

パーミッション

DBA 権限。

関連する動作

なし

参照

- 「アンロードユーティリティを使用したデータの再構築」『SQL Anywhere サーバー SQL の使用法』
- 「データベースの暗号化と復号化」『SQL Anywhere サーバー データベース管理』
- 「テーブル暗号化」『SQL Anywhere サーバー データベース管理』
- 「単純暗号化の使用」『SQL Anywhere サーバー データベース管理』
- 「強力な暗号化」『SQL Anywhere サーバー データベース管理』
- 「CREATE DECRYPTED DATABASE 文」530 ページ
- 「CREATE ENCRYPTED FILE 文」537 ページ
- 「CREATE DECRYPTED FILE 文」531 ページ
- 「CREATE DATABASE 文」521 ページ
- 「ALTER TABLE 文」462 ページ
- 「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、サンプルデータベースの暗号化されたコピー *demoEnc.db* を作成します。新しいデータベースの暗号化には、AES256 暗号化が使用されます。

```
CREATE ENCRYPTED DATABASE 'demoEnc.db'
  FROM 'demo.db'
  KEY 'Sd8f6654*Mnn'
  ALGORITHM 'AES256';
```

次の例は、サンプルデータベースのコピー *demoTableEnc.db* を作成します。新しいデータベースでは、テーブル暗号化が有効になります。アルゴリズムの指定なしでキーが指定されているため、AES 暗号化が使用されます。

```
CREATE ENCRYPTED TABLE DATABASE 'demoTableEnc.db'
  FROM 'demo.db'
  KEY 'Sd8f6654';
```

CREATE ENCRYPTED FILE 文

データベースファイル、トランザクションログ、トランザクションログミラー、または DB 領域の、強力な暗号化が施されたコピーを作成します。

構文

CREATE ENCRYPTED FILE *newfile*
FROM *oldfile*
{ **KEY** *key* | **KEY** *key* **OLD KEY** *oldkey* }
[**ALGORITHM** {
  'AES'
  | **'AES256'**
  | **'AES_FIPS'**
  | **'AES256_FIPS'** }]

パラメーター

FROM 句　CREATE ENCRYPTED FILE 文を実行する既存ファイル (*oldfile*) の名前を指定します。

KEY 句　使用する暗号化キーを指定します。

OLD KEY 句　ファイルの暗号化に使用する現在のキーを指定します。

ALGORITHM 句　ファイルの暗号化に使用されるアルゴリズムを指定します。アルゴリズムを指定しない場合は、デフォルトで AES (128 ビットの暗号化) が使用されます。

備考

この文は、データベースでリカバリが必要であり、データベースの暗号化されたコピーをサポート目的で必要な場合に使用します。この文の使用は、トランザクションログファイル、トランザクションログミラーファイル、DB 領域ファイルなどのデータベース関連ファイルの暗号化でも必要です。

暗号化ファイルを作成しているデータベースに接続できません。別のデータベースに接続されている必要があります。

データベース関連ファイルを暗号化するときは、データベースのすべての関連ファイルに同じアルゴリズムとキーを指定する必要があります。

関連する DB 領域またはトランザクションログファイルが *oldfile* に存在し、これらも暗号化する場合は、該当するファイルの新しい名前とロケーションを新しいデータベースに格納する必要があります。そのためには、次の手順に従います。

- 新しいデータベースに対して dblog -t を実行して、トランザクションログの名前とロケーションを変更します。
- 新しいデータベースに対して dblog -m を実行して、トランザクションログミラーの名前とロケーションを変更します。
- 新しいデータベースに対して ALTER DBSPACE 文を実行して、DB 領域ファイルのロケーションと名前を変更します。

この文を使用して、データベースの暗号化アルゴリズムとキーを変更できます。ただし、CREATE ENCRYPTED FILE 文は、新しいファイル (*newfile*) を作成しますが、前バージョンのファイル (*oldfile*) を置換または削除することはありません。

このプロセスでトランザクションログファイルの名前が変わることはありません。したがって、データベースとトランザクションログファイルの名前を変更した場合は、復号化後のデータベースに対して dblog -t を実行する必要があります。

dbunload -an オプションに -ek または -ep を使用してデータベースのアンロードと再ロードを行うと、既存のデータベースを暗号化したり、既存の暗号化キーを変更したりできます。

テーブルの暗号化が有効なデータベースがある場合、この文を使用してデータベースを暗号化することはできません。ただし、この文を使用して、テーブルの暗号化に使用したキーを変更する

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ことはできます。テーブル暗号化が有効になっているデータベースを暗号化するには、CREATE ENCRYPTED DATABASE 文を使用します。

この文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

> **注意**
> FIPS 認定の暗号化をサポートしないプラットフォームもあります。サポートされるプラットフォームのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

パーミッション

DBA 権限が必要です。

Windows Mobile では、ARM プロセッサー用に AES_FIPS および AES256_FIPS アルゴリズムのみがサポートされています。

関連する動作

なし

参照

- 「CREATE ENCRYPTED DATABASE 文」535 ページ
- 「データベースの暗号化と復号化」『SQL Anywhere サーバー データベース管理』
- 「CREATE ENCRYPTED DATABASE 文」535 ページ
- 「CREATE DECRYPTED FILE 文」531 ページ
- 「CREATE DECRYPTED DATABASE 文」530 ページ
- 「アンロードユーティリティ (dbunload)」『SQL Anywhere サーバー データベース管理』
- 「トランザクションログユーティリティ (dblog)」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、contacts データベースを暗号化し、AES_FIPS で暗号化された新しいデータベース contacts2 を作成します。

```
CREATE ENCRYPTED FILE 'contacts2.db'
FROM 'contacts.db'
   KEY 'Sd8f6654*Mnn'
   ALGORITHM 'AES_FIPS';
```

次の例は、contacts データベースと contacts トランザクションログファイルを暗号化して、両ファイルの名前を変更します。トランザクションログの名前が変更されていてもデータベースファイルは引き続き古いログを指しているため、dblog -ek Sd8f6654*Mnn -t contacts2.log contacts.db を実行する必要があります。

```
CREATE ENCRYPTED FILE 'contacts2.db'
   FROM 'contacts.db'
   KEY 'Sd8f6654*Mnn';
CREATE ENCRYPTED FILE 'contacts2.log'
   FROM 'contacts.db'
   KEY 'Sd8f6654*Mnn';
```

次の例は、contacts データベースと contacts トランザクションログファイルを暗号化して、元のトランザクションログファイル名は変更しません。この場合は、ファイル名が変更されないので、dblog を実行する必要はありません。

```
CREATE ENCRYPTED FILE 'newpath¥contacts.db'
  FROM 'contacts.db'
  KEY 'Sd8f6654*Mnn';
CREATE ENCRYPTED FILE 'newpath¥contacts.log'
  FROM 'contacts.log'
  KEY 'Sd8f6654*Mnn';
```

データベースの暗号化キーを変更するには、次の文に示すように最初に新しいキーを使用してデータベースファイルのコピーを作成します。

```
CREATE ENCRYPTED FILE 'newcontacts.db'
 FROM 'contacts.db'
 KEY 'newkey' OLD KEY 'oldkey';
```

暗号化されたファイルを作成した後、*contacts.db* を削除し、次に *newcontacts.db* の名前を *contacts.db* に変更します。

CREATE EVENT 文

イベントとイベントに関連付けて定義済みアクションを自動化するイベントハンドラーを定義し、スケジュールされたアクションを定義します。

構文

CREATE EVENT [*owner*.]*event-name*
[**TYPE** *event-type*
  [**WHERE** *trigger-condition* [**AND** *trigger-condition*] ...]
  | **SCHEDULE** *schedule-spec*, ...]
[**ENABLE** | **DISABLE**]
[**AT** { **CONSOLIDATED** | **REMOTE** | **ALL** }]
[**HANDLER**
  BEGIN
...
  END]

event-type :
 BackupEnd
| **Connect**
| **ConnectFailed**
| **DatabaseStart**
| **DBDiskSpace**
| **Deadlock**
| **"Disconnect"**
| **GlobalAutoincrement**
| **GrowDB**
| **GrowLog**
| **GrowTemp**
| **LogDiskSpace**
| **MirrorFailover**
| **MirrorServerDisconnect**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| **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*

パラメーター

CREATE EVENT 句　イベント名は識別子です。イベントには作成者が関連付けられます。これはイベントを作成したユーザーであり、イベントハンドラーはその作成者のパーミッションで実行されます。これはストアドプロシージャーの実行と同じです。他のユーザーが所有するイベントを作成することはできません。

TYPE 句　オプションの WHERE 句と一緒に TYPE 句を指定するか、または SCHEDULE を指定できます。

event-type は、リストされたシステム定義のイベントタイプの 1 つです。イベントタイプでは大文字と小文字を区別しません。*event-type* がイベントをトリガーする条件を指定するには、WHERE 句を使用します。

○ **DiskSpace イベントタイプ**　データベースに DiskSpace タイプの 1 つに対応するイベントハンドラーがある場合、データベースサーバーは、使用するファイルに対応する各デバイスの空き領域を 30 秒おきにチェックします。

データベースが別々のドライブに複数の DB 領域を持っている場合、DBDiskSpace は各ドライブをチェックし、その中で最小の空き領域に基づいて動作します。

LogDiskSpace イベントタイプは、トランザクションログのロケーションと、トランザクションログミラーがあればそのロケーションをチェックし、最小の空き領域に基づいてレポートします。

DiskSpace イベントタイプは、Windows Mobile ではサポートされていません。

TempDiskSpace イベントタイプは、テンポラリディスク領域の容量をチェックします。

適切なイベントハンドラーが定義されている場合 (DBDiskSpace、LogDiskSpace、または TempDiskSpace)、データベースサーバーは、データベースファイルに対応する各デバイスの空き領域を 30 秒おきにチェックします。同様に、システムイベントタイプ ServerIdle を処理するようにイベントが定義されている場合、データベースサーバーは、前の 30 秒間に要求が処理されなかったときにハンドラーを通知します。

データベースサーバーの起動時に -fc オプションを指定して、データベースサーバーでファイルシステムがいっぱいになった場合のコールバック関数を実装できます。

○ **GlobalAutoincrement イベントタイプ**　このイベントは、GLOBAL AUTOINCREMENT の残りの値の数がその範囲の終わり 1% 未満になったときに、**それぞれの**挿入に対して発生します。このハンドラーの一般的なアクションは、このイベントのパラメーターとして指定されたテーブルと残りの値の数に基づいて、global_database_id オプションの新しい値を要求することです。

event_condition 関数と RemainingValues をこのイベントタイプの引数として使用できます。

○ **ServerIdle イベントタイプ**　データベースに ServerIdle タイプのイベントハンドラーがある場合、データベースサーバーは 30 秒おきにサーバーのアクティビティをチェックします。

○ **データベースミラーリングのイベントタイプ**　MirrorServerDisconnect イベントは、プライマリデータベースサーバーからミラーサーバーまたは監視サーバーへの接続が失われたときに発生します。MirrorFailover イベントは、サーバーがデータベースの所有権を取得したときに発生します。

WHERE 句　トリガー条件は、イベントが起動する条件を決定します。たとえば、トランザクションログが書き込まれるディスクの使用率が 80% を超えたときにアクションを実行する場合は、次のようなトリガー条件を使用します。

```
...
WHERE event_condition( 'LogDiskSpacePercentFree' ) < 20
...
```

event_condition 関数には、イベントタイプに有効な引数を指定してください。

複数の AND 条件を使って WHERE 句を構成できますが、OR 条件やその他の条件は使用できません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SCHEDULE 句　この句は、スケジュールされたアクションをいつ実行するかを指定します。時刻のシーケンスは、イベントハンドラーに定義された関連するアクションのトリガー条件セットとして動作します。

1 つのイベントとそのハンドラーに対し、複数のスケジュールを作成できます。これにより、複雑なスケジュールを実装できます。複数のスケジュールがある場合は、*schedule-name* を必ず指定しなければなりませんが、スケジュールが 1 つしかない場合は、*schedule-name* を省略できます。

スケジュールされたイベントの定義に EVERY または ON が含まれる場合、そのイベントは反復されます。このどちらの予約語も使用されていない場合、イベントは最大でも 1 回しか実行されません。反復されないスケジュールイベントを作成する場合に、そのスケジュールの開始時刻が過ぎているときは、エラーになります。反復されないスケジュールされたイベントが経過すると、スケジュールは削除されますが、イベントハンドラーは削除されません。

スケジュールされたイベントの時刻は、スケジュールの作成時に計算され、イベントハンドラーの実行が完了したときに再計算されます。次のイベント時刻を計算するときには、イベントのスケジュールが調べられ、起動時刻が現在以降のスケジュールから次のスケジュール時刻が決定されます。9:00 から 5:00 の間に 1 時間ごとに実行され、実行に 65 分を要するイベントハンドラーは、9:00、11:00、1:00、3:00、5:00 に実行されます。実行を重複させる場合は、複数のイベントを作成します。

スケジュール定義に使用する句は次のとおりです。

- **START TIME 句**　イベントがスケジュールされた各日の最初のスケジュール時刻。*start-time* パラメーターは文字列であり、変数や NOW() などの式にはできません。START DATE を指定した場合、START TIME はその日付とそれ以降の毎日 (スケジュールに EVERY または ON が含まれる場合) を表します。START DATE を指定しない場合、START TIME は現在の日付 (時刻が経過していない場合) とそれ以降の毎日 (スケジュールに EVERY または ON が含まれる場合) となります。START TIME *start-time* 句は BETWEEN *start-time* AND '23:59:59' と同義です。

- **BETWEEN...AND 句**　1 日のうち、スケジュールされた時刻が発生する範囲。*start-time* パラメーターと *end-time* パラメーターは文字列であり、変数や NOW() などの式にはできません。START DATE を指定した場合、スケジュールされた時刻はその日が来るまで発生しません。

- **EVERY 句**　連続してスケジュールするときのイベント発生の間隔。スケジュールされたイベントは、その日の START TIME より後、または BETWEEN...AND で指定した範囲内でのみ発生します。

- **ON 句**　スケジュールされたイベントが発生する日のリスト。EVERY を指定した場合、デフォルトは毎日です。これらは曜日または日付として指定できます。

  曜日は、Mon、Tues のようになります。Monday のような完全形も使用できます。使用言語が、英語でない場合、接続文字列でクライアントによって要求された言語でない場合、データベースサーバーメッセージウィンドウに表示される言語でない場合は、完全形を使用します。

  日付は、0 ～ 31 の整数で指定します。0 は月の末日を表します。

- **START DATE 句**　スケジュールされたイベントが開始される日付。この値は文字列であり、変数や TODAY() などの式にはできません。デフォルトは現在の日付です。

  スケジュールされたイベントハンドラーが完了するたびに、イベントに対してスケジュールされた次の時刻と日付を計算するために次のアクションが実行されます。

  1. EVERY 句を使用した場合は、次のスケジュールされた時刻が現在の日付にあり、BETWEEN...AND 句で指定された範囲の終わりより前であるかどうかを調べます (範囲が指定された場合)。そうであれば、これが次のスケジュールされた時刻となります。

  2. 次のスケジュールされた時刻が現在の日付にない場合は、イベントが実行される次の日付を調べ、その日付の START TIME、または BETWEEN...AND で指定された範囲の始まりを確認します。

- **ENABLE | DISABLE 句**　イベントハンドラーはデフォルトで有効になっています。DISABLE を指定すると、スケジュールされた時刻に達したりトリガー条件が発生しても、イベントハンドラーは起動しません。TRIGGER EVENT 文は、無効に設定されているイベントハンドラーを起動**しません**。

- **AT 句**　SQL Remote 設定時にリモートデータベースまたは統合データベースで AT 句を使用してイベントを処理するデータベースを制限する場合にのみ、この句を使用してください。

  SQL Remote のイベント作成時に AT 句を使用しなかった場合は、すべてのデータベースでイベントが実行されます。統合データベースで実行される場合、この文はすでに抽出されているリモートデータベースには影響しません。

- **HANDLER 句**　各イベントが 1 つのハンドラーを持ちます。

備考

イベントは次の操作に使用できます。

- **アクションのスケジュール**　データベースサーバーは、時刻指定されたスケジュールに沿ってアクションを実行します。この機能を使用すると、バックアップ、妥当性検査、レポートテーブルへのデータ追加に使用するクエリなどのタスクをスケジュールして実行できます。

- **イベント処理アクション**　データベースサーバーは、事前に定義されたイベントが発生するとアクションを実行します。この機能を使用すると、ディスク領域の使用量が指定の割合を超えたときのディスク領域の制限などのタスクを、スケジュールして実行できます。実行中にエラーが検出されなかった場合は、イベントハンドラーアクションがコミットされ、エラーが検出された場合は、ロールバックされます。

イベント定義は 2 つの部分からなります。トリガー条件とは、ディスク領域の使用量が指定のスレッショルドを超えるなどの出来事をいいます。スケジュールとは、時刻のセットのことで、それぞれの時刻がトリガー条件の役割を果たします。トリガー条件が満たされると、イベントハンドラーが実行されます。イベントハンドラーには、複合文 (BEGIN ...END) の中に指定された 1 つまたは複数のアクションが含まれています。

トリガー条件やスケジュールを指定しない場合は、明示的な TRIGGER EVENT 文だけがイベントをトリガーします。開発段階では TRIGGER EVENT を使ってイベントハンドラーをテストし、テストの完了後にスケジュールまたは WHERE 句を追加することができます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

イベントエラーはデータベースサーバーメッセージログに出力されます。

イベントハンドラーが実行されるたびに、エラーが発生しなかった場合、COMMIT が発生します。エラーが生成された場合は、ROLLBACK が発生します。

イベントハンドラーがトリガーされると、データベースサーバーは event_parameter 関数を使用して、イベントをトリガーした接続 ID などのコンテキスト情報をイベントハンドラーに渡します。

パーミッション

DBA 権限。

イベントハンドラーは、イベント所有者のパーミッションに基づいて別の接続上で実行されます。DBA 以外の権限で実行するには、イベントハンドラーからプロシージャーを呼び出します。プロシージャーは、その所有者のパーミッションを使用して実行されます。イベントハンドラーが実行される接続は、パーソナルデータベースサーバーの最大接続数である 10 には含められません。

関連する動作

オートコミット。

参照

- 「BEGIN 文」497 ページ
- 「ALTER EVENT 文」417 ページ
- 「COMMENT 文」512 ページ
- 「DROP EVENT 文」713 ページ
- 「TRIGGER EVENT 文」960 ページ
- 「EVENT_PARAMETER 関数 [システム]」223 ページ
- 「EVENT_CONDITION 関数 [システム]」222 ページ
- 「-fc dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「システムイベント」『SQL Anywhere サーバー データベース管理』
- 「イベントのトリガー条件」『SQL Anywhere サーバー データベース管理』
- 「データベースサーバー動作のロギング」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリングにおけるシステムイベント」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

データベース・サーバに対して、毎日午前 1 時に最初のテープ・ドライブを使用してテープへの自動バックアップを実行するように指示します。

```
CREATE EVENT DailyBackup
SCHEDULE daily_backup
START TIME '1:00AM' EVERY 24 HOURS
HANDLER
  BEGIN
    BACKUP DATABASE TO '¥¥¥¥.¥¥tape0'
```

```
    ATTENDED OFF
  END;
```

データベースサーバーに対して、月曜日から金曜日までの午前 8 時から午後 6 時まで、1 時間ごとにトランザクションログのみの自動バックアップを実行するように指示します。

```
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;
```

次にスケジュールされているイベントの実行時期を決定します。

```
SELECT DB_EXTENDED_PROPERTY( 'NextScheduleTime', 'HourlyLogBackup');
```

CREATE EXISTING TABLE 文

リモートサーバー上の既存のオブジェクトを表す新しいプロキシテーブルを作成します。

構文

CREATE EXISTING TABLE [*owner*.]*table-name*
[*column-definition*, ...]
AT *location-string* | *variable-name*

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*

パラメーター

AT 句　AT 句は、リモートオブジェクトのロケーションを指定します。AT 句は、デリミターとしてセミコロン (;) をサポートします。セミコロンが *location-string* 文字列のどこかにある場合、そのセミコロンはフィールドデリミターです。セミコロンがない場合は、ピリオドがフィールドデリミターです。この動作により、データベースと所有者の各フィールドにファイル名と拡張子を使用できます。たとえば、次の文は、テーブル a1 を Microsoft Access ファイル *mydbfile.mdb* にマッピングします。

```
CREATE EXISTING TABLE a1
AT 'access;d:¥mydbfile.mdb;;a1';
```

プロキシテーブルを使用する場合、*variable-name* は SQL 変数 *variable-name* の内容に置き換えられます。これは、CHAR、VARCHAR、LONG VARCHAR 型でなければなりません。AT 句の変数の使用の詳細については、「ディレクトリアクセスサーバーの作成 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』の例 2 を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

CREATE EXISTING TABLE 文は、外部のロケーションにあるテーブルに対応する、新しいローカルのプロキシテーブルを作成します。CREATE EXISTING TABLE 文は、CREATE TABLE 文の変形です。EXISTING キーワードを CREATE TABLE で使用すると、テーブルがすでにリモートに存在していて、そのメタデータが SQL Anywhere にインポートされるように指定できます。これにより、SQL Anywhere ユーザーにとって可視のエンティティであるように、リモートテーブルが設定されます。SQL Anywhere は、テーブルを作成する前に、外部ロケーションにテーブルが存在するかどうかを確認します。

オブジェクト (ホストデータファイルまたはリモートサーバーオブジェクトのどちらかとして) が存在しない場合、この文は拒否されてエラーメッセージが出力されます。

ホストデータファイルまたはリモートサーバーテーブルのインデックス情報が抽出され、システムテーブル ISYSIDX のローを作成するために使用されます。この情報により、サーバー関係のインデックスとキーが定義されて、クエリオプティマイザーがこのテーブルに存在する可能性のあるすべてのインデックスを考慮できるようになります。

参照整合性制約は、必要に応じてリモートロケーションに渡されます。

column-definitions が指定されていない場合、SQL Anywhere は、リモートテーブルから取得するメタデータからカラムリストを引き出します。*column-definitions* が指定されている場合、SQL Anywhere は、その *column-definitions* を検証します。カラム名、データ型、長さ、IDENTITY プロパティ、NULL プロパティについて、次の点がチェックされます。

- カラム名が一致しなければなりません (大文字小文字は無視されます)。
- CREATE EXISTING TABLE 文のデータ型は、リモートロケーションのカラムのデータ型と一致するか、またはそのデータ型に変換可能でなければなりません。たとえば、ローカルカラムのデータ型が通貨として定義されているのに対して、リモートカラムのデータ型が数値である場合があります。
- 各カラムの NULL プロパティがチェックされます。ローカルカラムの NULL プロパティがリモートカラムの NULL プロパティと同じではないと、警告メッセージが出力されますが、文はアボートしません。
- 各カラムの長さがチェックされます。CHAR、VARCHAR、BINARY、VARBINARY、DECIMAL、NUMERIC の各カラムの長さが一致しない場合は、警告メッセージが出力されますが、コマンドはアボートしません。

  CREATE EXISTING 文には、実際のリモートカラムリストのサブセットだけをインクルードすることができます。

パーミッション

RESOURCE 権限が必要です。別のユーザーのテーブルを作成するには、DBA 権限が必要です。

Windows Mobile ではサポートされません。

関連する動作

オートコミット。

参照

- 「CREATE TABLE 文」651 ページ
- 「プロキシテーブルのロケーション」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　Adaptive Server Enterprise によってサポートされます。*location-string* のフォーマットは実装依存です。

例

リモートサーバー server_a にある blurbs テーブルのプロキシテーブル blurbs を作成します。

```
CREATE EXISTING TABLE blurbs
( author_id ID not null,
copy text not null)
AT 'server_a.db1.joe.blurbs';
```

リモートサーバー server_a にある blurbs テーブルに対して blurbs という名前のプロキシテーブルを作成します。SQL Anywhere は、リモートテーブルから取得したメタデータからカラムリストを引き出します。

```
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs';
```

SQL Anywhere のリモートサーバー demo12 にある Employees テーブルのプロキシテーブル rda_employees を作成します。

```
CREATE EXISTING TABLE rda_employees
AT 'demo12...Employees';
```

CREATE EXTERNLOGIN 文

リモートサーバーとの通信に使用される代替ログイン名とパスワードを割り当てます。

構文

CREATE EXTERNLOGIN *login-name*
TO *remote-server*
[**REMOTE LOGIN** *remote-user* [**IDENTIFIED BY** *remote-password*]]

パラメーター

login-name　ローカルユーザーログイン名を指定します。統合化ログインを使用する場合、*login-name* は Windows ユーザーまたはグループにマッピングされたデータベースユーザーです。

TO 句　リモートサーバーの名前を指定します。

REMOTE LOGIN 句　REMOTE LOGIN 句は、ローカルユーザー *login-name* に対して、*remote-server* 上にユーザーアカウントを指定します。REMOTE LOGIN 句の値は 128 バイトまでに制限されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

IDENTIFIED BY 句　IDENTIFIED BY 句は、*remote-password* が *remote-user* のパスワードになるように指定します。*remote-user* と *remote-password* の組み合わせは remote-server において有効でなければなりません。

IDENTIFIED BY 句を省略すると、NULL のパスワードがリモートサーバーに送信されます。ただし、IDENTIFIED BY "" (空の文字列) を指定すると、空の文字列がパスワードとして送信されます。

備考

デフォルトでは、SQL Anywhere は、クライアントに代わってリモートサーバーに接続するときは、常にそのクライアントの名前とパスワードを使用します。CREATE EXTERNLOGIN は、リモートサーバーとの通信に使用される代替ログイン名とパスワードを割り当てます。

REMOTE LOGIN 句が必要なのは、リモートサーバーが接続にユーザー ID とパスワードを必要とする場合のみです。リモートログインなしの外部ログインがあると、DBA はリモートサーバーにアクセスできるユーザーを制御できます。また、リモートアクセスレイヤーに対し、リモートサーバーへのログインにユーザー ID とパスワードが不要であることを指示できます。たとえば、ディレクトリアクセスサーバークラスには、ディレクトリサーバーへのアクセスを制限するために外部ログインが必要ですが、ディレクトリサーバーはユーザー ID とパスワードの検証を行わないため、リモートログインは不要です。

パスワードは内部的に暗号化された形式で保存されます。*remote-server* をローカルサーバーに認識させるには、ISYSSERVER テーブルのエントリを指定します。

パスワードの自動有効期限を使用するサイトでは、外部ログインのために、定期的なパスワードの更新を計画してください。

CREATE EXTERNLOGIN は、トランザクション内からは使用できません。

パーミッション

login-name の外部ログインを追加または修正できるのは、DBA 権限を持つユーザーのみです。

Windows Mobile ではサポートされません。

関連する動作

オートコミット。

参照

- 「DROP EXTERNLOGIN 文」713 ページ
- 「CREATE SERVER 文」618 ページ
- 「外部ログインの作成」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

サーバー sybase1 に接続するときに、ローカルユーザー DBA をパスワード Plankton が設定されたユーザー sa にマッピングします。

```
CREATE EXTERNLOGIN DBA
TO sybase1
REMOTE LOGIN sa
IDENTIFIED BY Plankton;
```

CREATE FUNCTION 文 [外部呼び出し]

ネイティブ関数または外部関数へのインターフェイスを作成します。

構文

CREATE [**OR REPLACE**] **FUNCTION** [*owner*.]*function-name*
([*parameter*, ...] **)**
RETURNS *data-type*
[**SQL SECURITY** { **INVOKER** | **DEFINER** }]
[[**NOT**] **DETERMINISTIC**]
EXTERNAL NAME *external-call* [**LANGUAGE** *environment-name*]

parameter :
[**IN**] *parameter-name data-type* [**DEFAULT** *expression*]

environment-name :
C_ESQL32
| **C_ESQL64**
| **C_ODBC32**
| **C_ODBC64**
| **CLR**
| **JAVA**
| **PERL**
| **PHP**

パラメーター

パラメーター名は、データベース識別子に対するルールに従って付けてください。パラメーター名は、有効な SQL データ型にします。また、キーワード IN のプレフィクスを付けて、引数が関数に値を提供する式であることを示してください。ただし、関数パラメーターはデフォルトで IN です。

関数を実行するときに、すべてのパラメーターを指定する必要はありません。CREATE FUNCTION 文の中に DEFAULT 値がある場合、不明のパラメーターにデフォルト値を割り当てます。呼び出すときに引数を指定せず、デフォルトも設定されていない場合には、エラーが発生します。

OR REPLACE 句　OR REPLACE を指定すると、新しい関数が作成されるか、同じ名前の既存の関数が置き換えられます。この句によって、関数の定義は変更されますが、既存のパーミッションは保持されます。OR REPLACE 句をテンポラリ関数で使用することはできません。

RETURNS 句　RETURNS 句は、関数の結果のデータ型を指定するときに使用します。

SQL SECURITY 句　SQL SECURITY 句は、INVOKER (関数を呼び出すユーザー) または DEFINER (関数を所有するユーザー) として関数が実行されるかどうかを定義します。デフォルトは DEFINER です。外部呼び出しの場合、この句は、外部環境での修飾されていないオブジェクト参照に対して所有者のコンテキストを確立します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SQL SECURITY INVOKER が指定されている場合、関数を呼び出すユーザーごとに注釈を行う必要があるため、メモリ使用量が増えます。また、SQL SECURITY INVOKER が指定されている場合は、呼び出し側としても名前の決定が行われます。このため、すべてのオブジェクト名 (テーブル、プロシージャーなど) を該当する所有者で修飾する場合は注意が必要です。たとえば、user1 が次の関数を作成するとします。

```
CREATE FUNCTION user1.myFunc()
  RETURNS INT
  SQL SECURITY INVOKER
  BEGIN
    DECLARE res INT;
    SELECT COUNT(*) INTO res FROM table1;
    RETURN res;
  END;
```

user2 がこの関数を実行しようとし、テーブル user2.table1 が存在**しない**場合、テーブルルックアップエラーが生じます。さらに、user2.table1 が存在**する**場合は、意図する user1.table1 の代わりにこのテーブルが使用されます。このような状況を防ぐには、文においてテーブル参照を修飾します (単なる table1 ではなく、user1.table1 とします)。

[NOT] DETERMINISTIC 句　この句は、関数が決定的か非決定的かを示すために使用します。この句が省略されると、関数の決定的な動作は指定されません (デフォルト)。

DETERMINISTIC と宣言された関数は、同じパラメーターセットで呼び出されるたび、同じ値を返します。

NOT DETERMINISTIC と宣言された関数は、同じパラメーターセットに対して同じ値を返すとはかぎりません。NOT DETERMINISTIC として宣言された関数は、クエリで呼び出されるたびに再評価されます。特定のパラメーターセットに対して関数が返す結果が変化するとわかっている場合は、この句を使用してください。

また、基本となるデータの修正などの関連する動作を伴う関数は、NOT DETERMINISTIC として宣言してください。たとえば、プライマリキー値を生成し、INSERT...SELECT 文で使用する関数は、次のように NOT DETERMINISTIC として宣言してください。

```
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;
```

特定の入力パラメーターに対して常に同じ値を返す関数は、DETERMINISTIC として宣言できます。

EXTERNAL NAME *native-call* 句

native-call :
[*operating-system*:]*function-name*@*library*; ...

operating-system : **Unix**

> **注意**
> EXTERNAL NAME 句は TEMPORARY 関数ではサポートされていませんので、注意してください。

LANGUAGE 属性なしで EXTERNAL NAME 句を使用する関数は、C などのプログラミング言語で記述されたネイティブ関数へのインターフェイスを定義します。ネイティブ関数は、データベースサーバーによってそのアドレス領域にロードされます。

library 名にはファイル拡張子を付けることができます。これは通常、Windows では *.dll*、UNIX では *.so* です。拡張子がない場合、ライブラリに対するプラットフォーム固有のデフォルトのファイル拡張子が追加されます。次に、正式な例を示します。

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
```

上記の EXTERNAL NAME 句を、プラットフォーム固有のデフォルト値を使用して簡単に記述すると、次のようになります。

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib';
```

呼び出されると、関数を含むライブラリがデータベースサーバーのアドレス領域にロードされます。ネイティブ関数は、サーバーの一部として実行されます。この場合、関数が原因で障害が発生すると、データベースサーバーは終了されます。したがって、外部環境での関数のロードと実行には、LANGUAGE 属性を使用することをおすすめします。関数が原因で外部環境に障害が発生しても、データベースサーバーは実行し続けます。

EXTERNAL NAME *c-call* LANGUAGE {C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 } 句　コンパイル済みネイティブ C 関数をデータベースサーバー内でなく外部環境で呼び出すには、ストアドプロシージャーまたはファンクションを EXTERNAL NAME 句で定義し、後続の LANGUAGE 属性で C_ESQL32、C_ESQL64、C_ODBC32、C_ODBC64 のいずれか 1 つを指定します。

LANGUAGE 属性が指定されると、その関数を含むライブラリが外部プロセスによってロードされ、外部関数がその外部プロセスの一部として実行されます。この場合、関数が原因で障害が発生しても、データベースサーバーは実行し続けます。

次に、関数定義の例を示します。

```
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* LANGUAGE CLR 句　.NET 関数を外部環境で呼び出すには、関数インターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE CLR 属性を指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CLR ストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファンクションのコードは C# または Visual Basic などの .NET 言語で記述され、その実行はデータベースサーバーの外側 (つまり別の .NET 実行ファイル内) で行われます。

次に、関数定義の例を示します。

```
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 )'
LANGUAGE CLR;
```

EXTERNAL NAME *perl-call* LANGUAGE PERL 句　Perl 関数を外部環境で呼び出すには、関数インターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE PERL 属性を指定します。

Perl ストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファンクションのコードは Perl で記述され、その実行はデータベースサーバーの外側 (つまり Perl 実行インスタンス内) で行われます。

次に、関数定義の例を示します。

```
CREATE FUNCTION PerlWriteToConsole( IN str LONG VARCHAR)
RETURNS INT
EXTERNAL NAME '<file=PerlConsoleExample>
   WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```

EXTERNAL NAME *php-call* LANGUAGE PHP 句　PHP 関数を外部環境で呼び出すには、関数インターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE PHP 属性を指定します。

PHP ストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファンクションのコードは PHP で記述され、その実行はデータベースサーバーの外側 (つまり PHP 実行インスタンス内) で行われます。

次に、関数定義の例を示します。

```
CREATE FUNCTION PHPPopulateTable()
RETURNS INT
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;
```

EXTERNAL NAME *java-call* LANGUAGE JAVA 句　Java メソッドを外部環境で呼び出すには、関数インターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE JAVA 属性を指定します。

Java とのインターフェイスとなるストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファ

ンクションのコードは Java で記述され、その実行はデータベースサーバーの外側 (つまり Java VM 内) で行われます。

次に、関数定義の例を示します。

```
CREATE FUNCTION HelloDemo( IN name LONG VARCHAR )
RETURNS INT
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

備考

CREATE FUNCTION 文はデータベースに関数を作成します。DBA 権限があるユーザーは、所有者を指定することによって他のユーザーの関数を作成できます。関数は、SQL 式の一部として呼び出されます。

複数の関数からテンポラリテーブルを参照する場合、テンポラリテーブル定義が矛盾していたり、テーブルを参照する文がキャッシュされていたりすると、問題が発生する可能性があります。

パーミッション

テンポラリ関数を作成するのでないかぎり、RESOURCE 権限が必要です。

外部関数を作成する場合、および別のユーザーの関数を作成する場合、DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「ALTER FUNCTION 文」421 ページ
- 「CALL 文」503 ページ
- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [Web サービス]」555 ページ
- 「CREATE PROCEDURE 文 [外部呼び出し]」583 ページ
- 「DROP FUNCTION 文」714 ページ
- 「GRANT 文」782 ページ
- 「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』
- 「SQL Anywhere 外部呼び出しインターフェイス」『SQL Anywhere サーバー プログラミング』
- 「ESQL 外部環境と ODBC 外部環境」『SQL Anywhere サーバー プログラミング』
- 「CLR 外部環境」『SQL Anywhere サーバー プログラミング』
- 「Java 外部環境」『SQL Anywhere サーバー プログラミング』
- 「PHP 外部環境」『SQL Anywhere サーバー プログラミング』
- 「PERL 外部環境」『SQL Anywhere サーバー プログラミング』
- 「SQL データ型」85 ページ
- 「プロシージャー内でのテンポラリテーブルの参照」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　外部言語環境の CREATE FUNCTION は SQL/2008 標準のコア機能ですが、SQL Anywhere でサポートされている一部のコンポーネントはオプションの SQL/2008 言語機能です。これらの機能のサブセットを次に示します。

○SQL SECURITY 句は、オプションの言語機能 T324 です。

○SQL 関数に LONG VARCHAR、LONG NVARCHAR、または LONG BINARY の値を渡す機能は、言語機能 T041 です。

○LANGUAGE JAVA のサポートは、オプションの SQL/2008 言語機能 J621 です。

○CREATE TABLE または DROP TRIGGER などの文を使用して外部関数内でスキーマオブジェクトを作成または変更する機能は、言語機能 T653 です。

○CONNECT、EXECUTE IMMEDIATE、PREPARE、および DESCRIBE などの文を使用して外部関数内で動的 SQL 文を使用する機能は、言語機能 T654 です。

CREATE FUNCTION 文のいくつかの句は、ベンダー拡張です。これらを以下に示します。

○LANGUAGES 句での C_ESQL32、C_ESQL64、C_ODBC32、C_ODBC64、CLR、PERL、PHP のサポートは、ベンダー拡張です。

○*external-call* のフォーマットは実装依存です。

○特定のルーチンパラメーターのオプションの DEFAULT 句は、ベンダー拡張です。

○オプションの OR REPLACE 句は、ベンダー拡張です。

- **Transact-SQL**　外部ルーチンに対する CREATE FUNCTION は、Adaptive Server Enterprise でサポートされています。Adaptive Server Enterprise では、外部関数の外部環境 (SQL/2008 言語機能 J621) として LANGUAGE JAVA のみがサポートされています。

CREATE FUNCTION 文 [Web サービス]

HTTP 要求を使用して HTTP 要求または SOAP 要求を行う Web クライアント関数を作成します。

構文

CREATE [**OR REPLACE**] **FUNCTION** [*owner*.]*function-name* **(** [*parameter*, ...] **)**
RETURNS *data-type*
URL *url-string*
[**HEADER** *header-string*]
[**SOAPHEADER** *soap-header-string*]
[**TYPE** {
  'HTTP[**:**{ **GET** | **POST**[**:***MIME-type*] | **PUT**[**:***MIME-type*] | **DELETE** | **HEAD** } **]'** |
  'SOAP[**:**{ **RPC** | **DOC** } **]'** }]
[**NAMESPACE** *namespace-string*]
[**CERTIFICATE** *certificate-string*]
[**CLIENTPORT** *clientport-string*]
[**PROXY** *proxy-string*]
[**SET** *protocol-option-string*]

url-string :
' { **HTTP** | **HTTPS** | **HTTPS_FIPS** }**:***//*[*user*:*password*@]*hostname*[**:***port*][*/path*]**'**

parameter :
[**IN**] *parameter-name data-type* [**DEFAULT** *expression*]

パラメーター

パラメーター名は、データベース識別子に対するルールに従って付けてください。パラメーター名は、有効な SQL データ型にします。また、キーワード IN のプレフィクスを付けて、引数が関数に値を提供する式であることを示してください。ただし、関数パラメーターはデフォルトで IN です。

関数を実行するときに、すべてのパラメーターを指定する必要はありません。CREATE FUNCTION 文の中に DEFAULT 値がある場合、不明のパラメーターにデフォルト値を割り当てます。呼び出すときに引数を指定せず、デフォルトも設定されていない場合には、エラーが発生します。

OR REPLACE 句　OR REPLACE を指定すると、新しい関数が作成されるか、同じ名前の既存の関数が置き換えられます。この句によって、関数の定義は変更されますが、既存のパーミッションは保持されます。OR REPLACE 句をテンポラリ関数で使用することはできません。

RETURNS 句　次のいずれかを指定して、SOAP 関数または HTTP 関数の戻り値のデータ型を定義します。

- CHAR
- VARCHAR
- LONG VARCHAR
- TEXT
- NCHAR
- NVARCHAR
- LONG NVARCHAR
- NTEXT
- XML
- BINARY
- VARBINARY
- LONG BINARY

返される値は、HTTP 応答の本文です。HTTP ヘッダー情報は含まれません。ステータス情報などの他の情報が必要な場合は、関数ではなくプロシージャーを使用します。

データ型は HTTP 応答の処理方法には影響しません。

URL 句　URL 句は HTTP または SOAP Web サービスクライアント関数を定義する場合にのみ使用します。URL 句は Web サービスの URI を指定します。オプションのユーザー名とパスワードのパラメーターは、HTTP 基本認証に必要なクレデンシャルとして機能します。HTTP 基本認証は、ユーザーとパスワードの情報を base-64 でエンコードし、HTTP 要求の Authentication ヘッダーに渡します。

HTTPS_FIPS を指定すると、強制的に FIPS 認定ライブラリが使用されます。HTTPS_FIPS を指定したときに、FIPS 認定ライブラリがない場合は、代わりに FIPS 認定ではないライブラリが使用されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

HTTP:GET タイプの関数の場合、URL 句内および (自動的に生成されて) プロシージャーに渡されたパラメーターからの両方でクエリパラメーターを指定できるようになりました。

```
URL 'http://localhost/service?parm=1
```

HEADER 句　HTTP Web サービスクライアント関数を作成する場合は、この句を使用して、HTTP 要求ヘッダーのエントリを追加または変更します。HTTP ヘッダーに指定できるのは印字可能な ASCII 文字のみで、大文字と小文字は区別されません。

SOAPHEADER 句　SOAP Web サービスを関数として宣言する場合は、この句を使用して 1 つ以上の SOAP 要求ヘッダーエントリを指定します。SOAP ヘッダーは、静的定数として宣言したり、代入パラメーターメカニズムを使用して動的に設定したりできます (hd1、hd2 などに IN、OUT、または INOUT パラメーターを宣言)。Web サービス関数では、1 つ以上の IN モード代入パラメーターを定義できますが、INOUT または OUT 代入パラメーターは定義できません。

TYPE 句　TYPE 句は Web サービス要求を行う場合に使用するフォーマットを指定します。SOAP が指定されている場合、または TYPE 句が含まれていない場合は、SOAP:RPC が使用されます。HTTP が指定されている場合は、HTTP:POST が使用されます。

TYPE 句では、HTTP:GET、HTTP:POST、HTTP:PUT の各タイプに MIME タイプを指定できます。*MIME タイプ*指定は、Content-Type 要求ヘッダーや操作モードの設定に使用することにより、1 つの呼び出しパラメーターのみで要求の本文を設定できます。パラメーターの置換後に Web サービスのストアド関数呼び出しを行う場合は、パラメーターがまったくなくなるか、1 つだけ残される場合があります。Web サービス関数を (置換後に) NULL 値またはパラメーターなしで呼び出すと、本文がなくコンテンツ長が 0 の要求になります。パラメーターの名前と値 (複数のパラメーターが可能) は、HTTP 要求の本文内で URL コード化されます。

一般的な MIME タイプの例を次に示します。

- text/plain
- text/html
- text/xml

TYPE 句のキーワードには次の意味があります。

- **HTTP:GET**　デフォルトでは、URL で指定されたパラメーターのエンコード用に、このタイプで MIME タイプ application/x-www-form-urlencoded が使用されます。

  たとえば、クライアントが URL http://localhost/WebServiceName?arg1=param1&arg2=param2 から要求を送信すると、次の要求が生成されます。

  ```
  GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
  // <End of Request - NO BODY>
  ```

- **HTTP:POST**　デフォルトでは、POST 要求の本文で指定されたパラメーターのエンコード用に、このタイプで MIME タイプ application/x-www-form-urlencoded が使用されます。URL パラメーターは、本文に格納されます。

  たとえば、クライアントが URL http://localhost/WebServiceName?arg1=param1&arg2=param2 から要求を送信すると、次の要求が生成されます。

```
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 は HTTP:POST と似ていますが、HTTP 要求メソッドが生成されます。HTTP:PUT タイプには、デフォルトのメディアタイプはありません。

次の例は、*put_data.sql* サンプルを実行している SQL Anywhere サーバーにデータをアップロードする、汎用のクライアントプロシージャーを設定する方法を示します。

```
ALTER 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**　サーバーにあるリソースを削除する Web サービスクライアントプロシージャーを設定できます。メディアタイプの指定はオプションです。

次の例は、*put_data.sql* サンプルを実行している SQL Anywhere サーバーからリソースを削除する、汎用のクライアントプロシージャーを設定する方法を示します。

```
ALTER PROCEDURE CDEL(resnm LONG VARCHAR, mediatype LONG VARCHAR)
   URL 'http://localhost/resource/!resnm'
   TYPE 'HTTP:DELETE:!mediatype';

CALL CDEL('hello', 'text/plain' );
```

○ **HTTP:HEAD**　HEAD メソッドは GET メソッドと同じですが、サーバーは本文を返しません。メディアタイプを指定できます。

```
ALTER PROCEDURE CHEAD(resnm LONG VARCHAR)
   URL 'http://localhost/resource/!resnm'
   TYPE 'HTTP:HEAD';

CALL CHEAD( 'hello' );
```

○ **SOAP:RPC**　このタイプは、Content-Type ヘッダーを 'text/xml' に設定します。SOAP 操作と SOAP パラメーターは、SOAP エンベロープの XML ドキュメントにカプセル化されます。

○ **SOAP:DOC**　このタイプは、Content-Type ヘッダーを 'text/xml' に設定します。SOAP:RPC タイプに似ていますが、より多くのデータ型を送信できます。SOAP 操作と SOAP パラメーターは、SOAP エンベロープの XML ドキュメントにカプセル化されます。

TYPE 句に MIME タイプを指定すると、その MIME タイプに Content-Type ヘッダーが自動的に設定されます。

NAMESPACE 句　SOAP クライアント関数にのみ適用されます。この句は、SOAP:RPC 要求と SOAP:DOC 要求の両方に通常必要なメソッドネームスペースを識別します。要求を処理する SOAP サーバーは、このネームスペースを使用して、SOAP 要求メッセージ本文内のエンティティの名前を解釈します。ネームスペースは、Web サービスサーバーから使用できる SOAP

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

サービスの WSDL (Web Services Description Language) から取得できます。デフォルト値は、関数の URL のオプションのパスコンポーネントの直前までです。

CERTIFICATE 句　安全な (HTTPS) 要求を行うには、HTTPS サーバーで使用される証明書にクライアントがアクセスできる必要があります。必要な情報は、セミコロンで区切られたキー／値のペアの文字列で指定されます。ファイルキーを使用して証明書のファイル名を指定したり、証明書キーを使用してサーバー証明書を文字列で指定したりできます。ファイルキーと証明書キーを一緒に指定することはできません。次のキーを使用できます。

キー	省略形	説明
file		証明書のファイル名
certificate	cert	証明書自体
company	co	証明書で指定された会社
unit		証明書で指定された会社の部署
name		証明書で指定された通称

証明書は、HTTPS サーバーに対する要求、または安全でないサーバーから安全なサーバーにリダイレクトされる可能性がある要求に対してのみ必要です。PEM でフォーマットされた証明書のみがサポートされています。

CLIENTPORT 句　HTTP クライアント関数が TCP/IP を使用して通信するポート番号を示します。この句は、ファイアウォールを介する通信のためのものであり、このような通信にのみおすすめします。ファイアウォールは TCP/UDP ポートに従ってフィルターします。単一のポート番号、ポート番号の範囲、または両方の組み合わせを指定できます。たとえば、CLIENTPORT '85,90-97' を指定できます。

PROXY 句　プロキシサーバーの URI を指定します。クライアントがプロキシを介してネットワークにアクセスする場合に使用します。この句は、関数がプロキシサーバーに接続し、そのプロキシサーバーを介して Web サービスに要求を送信することを示します。

SET 句　HTTP および SOAP のプロトコル固有の動作オプションを指定します。次の表に、サポートされている SET オプションを示します。CHUNK と VERSION は、HTTP プロトコルに適用されます。OPERATION は、SOAP プロトコルに適用されます。この句では、パラメーターの代入がサポートされています。

○ **'HTTP(CH[UNK]=*option*)'**　(HTTP または SOAP) このオプションでは、チャンクを使用するかどうかを指定します。チャンクを使用すると、HTTP メッセージを複数の部分に分割できます。可能な値は、ON (常にチャンクを使用)、OFF (チャンクは使用しない)、AUTO (内容が 8196 バイトを超えた場合にのみチャンクを使用、ただし自動生成されたマークアップを除く) です。たとえば、次の SET 句を使用すると、チャンクは有効になります。

```
SET 'HTTP(CHUNK=ON)'
```

CHUNK オプションを指定しないときの、デフォルトの動作は AUTO です。チャンクされた要求が AUTO モードのときに、ステータスが 505 (HTTP Version Not Supported) または 501

(Not Implemented) または 411 (Length Required) で失敗した場合、クライアントはチャンクされた転送コーディングなしで要求をリトライします。

チャンクされた転送コーディングによる要求を HTTP サーバーがサポートしない場合は、CHUNK オプションを OFF (チャンクを使用しない) に設定してください。

CHUNK モードは HTTP バージョン 1.1 からサポートされる転送エンコードです。CHUNK を ON に設定する場合、バージョン (VER) が 1.1 に設定される必要がありますが、デフォルトのバージョンとして 1.1 を使用する場合は、VER に何も設定しないでください。

○ **' HTTP(VER[SION]=*ver*)'** (HTTP または SOAP) このオプションでは、HTTP メッセージのフォーマットに使用する HTTP プロトコルのバージョンを指定します。たとえば、次の SET 句は HTTP のバージョンを 1.1 に設定します。

```
SET 'HTTP(VERSION=1.1)'
```

可能な値は 1.0 と 1.1 です。VERSION が指定されていない場合は、次のようになります。

●CHUNK が ON に設定された場合、1.1 が HTTP バージョンとして使用される

●CHUNK が OFF に設定された場合、1.0 が HTTP バージョンとして使用される

●CHUNK が AUTO に設定された場合、クライアントが CHUNK モードで送信しているかどうかによって 1.0 か 1.1 が使用される

○ **'REDIR(COUNT=*count*, STATUS=*status-list*)'** (HTTP または SOAP) **302 Found** および **303 See Other** などの HTTP 応答ステータスコードを使用して、特に HTTP POST を実行した後に Web アプリケーションを新しい URI にリダイレクトします。たとえば、クライアント要求は次のようにすることができます。

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```

web サーバー応答は次のようにすることができます。

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

応答では、クライアントは別の HTTP 要求を新しい URI に送信します。REDIR オプションを使用して、許可されるリダイレクションの最大数および自動的にリダイレクトする HTTP 応答ステータスコードを制御できます。

たとえば、SET 'REDIR(count=3, status=301,307)'では、最大制限の 3 リダイレクションおよび 301 と 307 ステータスのリダイレクションが可能です。302 や 303 などの他のリダイレクションステータスコードが受信されると、エラーが発効されます (SQLCODE -983)。

デフォルトのリダイレクション制限 *count* は 5 です。デフォルトでは、HTTP クライアントプロシージャーがすべての HTTP リダイレクションステータスコード (301、302、303、307) に応答して自動的にリダイレクトします。すべてのリダイレクションステータスコードを許可しないようにするには、SET REDIR(COUNT=0)を使用します。このモードでは、リダイレクション応答はエラーになりません (SQLCODE -983)。その代わりに、HTTP ステータスおよ

び応答ヘッダーによって結果セットが返されます。これにより、呼び出し側は **Location** ヘッダーに含まれる URI に基づいて、条件付きで要求を再発行できます。

303 See Other ステータスを受信する POST HTTP メソッドを指定する Web サービスプロシージャーは、GET HTTP メソッドを使用してリダイレクト要求を発行します。

Location ヘッダーには、絶対パスまたは相対パスのいずれかを含むことができます。どちらも HTTP クライアントプロシージャーによって処理されます。このヘッダーには、クエリパラメーターを含めることもでき、これらのパラメーターはリダイレクトされたロケーションに転送されます。たとえば、ヘッダーに次のようなパラメーターが含まれる場合、後続の GET または POST にこれらのパラメーターが含まれます。

```
Location: alternate_service?a=1&b=2
```

上の例では、クエリパラメーターは a=1&b=2 です。

○ **'SOAP(OP[ERATION]=*soap-operation-name*)'** (SOAP のみ) このオプションでは、SOAP 操作の名前を指定します (作成しているプロシージャーの名前と異なる場合)。OPERATION の値は、リモートプロシージャーコールの名前に似ています。たとえば、login という SOAP 操作を呼び出す accounts_login というプロシージャーを作成する場合は、次のように指定します。

```
CREATE FUNCTION accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
  SET 'SOAP(OPERATION=login)';
```

OPERATION オプションを指定しない場合、SOAP 操作の名前は、作成するプロシージャーの名前と一致する必要があります。

複数の *protocol-option* 設定を 1 つの SET 句で組み合わせると、次の文のようになります。

```
CREATE FUNCTION accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
  SET 'HTTP ( CHUNK=ON; VERSION=1.1 ), SOAP( OPERATION=login )'
  ...
```

備考

CREATE FUNCTION 文は、データベースに Web サービス関数を作成します。所有者名を指定すれば、別のユーザーに対する関数を作成できます。

パラメーター値は、要求の一部として渡されます。使用される構文は、要求のタイプによって決まります。HTTP:GET の場合、パラメーターは URL の一部として渡されます。HTTP:POST 要求の場合、値は要求の本文に置かれます。SOAP 要求へのパラメーターは、常に要求本文にバンドルされます。

パーミッション

RESOURCE 権限。

DBA 権限 (Java 関数を含む外部関数に必要)。

関連する動作

オートコミット。

参照

- 「SQL データ型」85 ページ
- 「チュートリアル：SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』
- 「HTTP 要求と SOAP 要求の構造」『SQL Anywhere サーバー プログラミング』
- 「HTTP 要求ヘッダーの管理」『SQL Anywhere サーバー プログラミング』
- 「ALTER FUNCTION 文」421 ページ
- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [外部呼び出し]」550 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「DROP FUNCTION 文」714 ページ
- 「RETURN 文」882 ページ
- 「Web サービスへの変数の指定」『SQL Anywhere サーバー プログラミング』
- 「チュートリアル：RAW サービスでの MIME タイプの操作」『SQL Anywhere サーバー プログラミング』
- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「Web クライアントアプリケーションの開発」『SQL Anywhere サーバー プログラミング』
- 「remote_idle_timeout オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　Adaptive Server Enterprise によってサポートされません。

例

次の文は、localhost 上で実行されている **get_picture** サービスからイメージを返す **cli_test1** という名前の関数を作成します。

```
CREATE FUNCTION cli_test1( image LONG VARCHAR )
RETURNS LONG BINARY
URL 'http://localhost/get_picture'
TYPE 'HTTP:GET';
```

次の文は、URL *http://localhost/get_picture?image=widget* の HTTP 要求を作成します。

```
SELECT cli_test1( 'widget' );
```

次の文は代入パラメーターを使用して、入力パラメーターとして要求 URL を渡すことができるようにします。CHUNK モードの転送エンコードをオフにするために、SET 句が使用されます。

```
CREATE FUNCTION cli_test2( image LONG VARCHAR, myurl LONG VARCHAR )
RETURNS LONG BINARY
URL '!myurl'
TYPE 'HTTP:GET'
SET 'HTTP(CH=OFF)'
HEADER 'ASA-ID';
```

次の文は、URL *http://localhost/get_picture?image=widget* の HTTP 要求を作成します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
CREATE VARIABLE a_binary LONG BINARY
a_binary = cli_test2('widget', 'http://localhost/get_picture');
SELECT a_binary;
```

CREATE FUNCTION 文

データベースにユーザー定義 SQL 関数を作成します。

構文

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*

parameter :
  [**IN**] *parameter-name data-type* [**DEFAULT** *expression*]

tsql-compound-statement :
sql-statement
sql-statement
 ...

パラメーター

パラメーター名は、データベース識別子に対するルールに従って付けてください。パラメーター名は、有効な SQL データ型にします。また、キーワード IN のプレフィクスを付けて、引数が関数に値を提供する式であることを示してください。ただし、関数パラメーターはデフォルトで IN です。

関数を実行するときに、すべてのパラメーターを指定する必要はありません。CREATE FUNCTION 文の中に DEFAULT 値がある場合、不明のパラメーターにデフォルト値を割り当てます。呼び出すときに引数を指定せず、デフォルトも設定されていない場合には、エラーが発生します。

OR REPLACE 句　CREATE OR REPLACE FUNCTION を指定すると、新しい関数が作成されるか、同じ名前の既存の関数が置き換えられます。関数が置き換えられた場合、関数の定義は変更されますが、既存のパーミッションは保持されます。

OR REPLACE 句をテンポラリ関数で使用することはできません。

TEMPORARY キーワード　CREATE TEMPORARY FUNCTION を指定すると、作成した接続でのみ参照できる関数になり、接続を削除すると関数も自動的に削除されます。テンポラリ関数を明示的に削除することもできます。テンポラリ関数に対して ALTER、GRANT、または REVOKE は実行できません。また他の関数とは異なり、テンポラリ関数はカタログやトランザクションログに記録されていません。

テンポラリ関数は、作成者 (現在のユーザー) または指定された所有者のパーミッションで実行されます。テンポラリ関数に所有者を指定できるのは次の場合です。

○テンポラリ関数が永続的なストアドプロシージャー内に作成された場合

○テンポラリ関数と永続的なストアドプロシージャーとで所有者が同じ場合

テンポラリ関数の所有者を削除するには、テンポラリ関数を先に削除する必要があります。

読み込み専用のデータベースに接続するときに、テンポラリ関数の作成と削除を行うことができます。

OR REPLACE 句をテンポラリ関数で使用することはできません。

RETURNS 句　RETURNS 句は、関数の結果のデータ型を指定するときに使用します。

SQL SECURITY 句　SQL SECURITY 句は、INVOKER (関数を呼び出すユーザー) または DEFINER (関数を所有するユーザー) として関数が実行されるかどうかを定義します。デフォルトは DEFINER です。

ON EXCEPTION RESUME 句　Transact-SQL のようなエラー処理を使用します。

[NOT] DETERMINISTIC 句　この句は、関数が決定的か非決定的かを示すために使用します。この句が省略されると、関数の決定的な動作は指定されません (デフォルト)。

DETERMINISTIC と宣言された関数は、同じパラメーターセットで呼び出されるたび、同じ値を返します。

NOT DETERMINISTIC と宣言された関数は、同じパラメーターセットに対して同じ値を返すとはかぎりません。NOT DETERMINISTIC として宣言された関数は、クエリで呼び出されるたびに再評価されます。特定のパラメーターセットに対して関数が返す結果が変化するとわかっている場合は、この句を使用してください。

また、基本となるデータの修正などの関連する動作を伴う関数は、NOT DETERMINISTIC として宣言してください。たとえば、プライマリキー値を生成し、INSERT...SELECT 文で使用する関数は、次のように NOT DETERMINISTIC として宣言してください。

```
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;
```

特定の入力パラメーターに対して常に同じ値を返す関数は、DETERMINISTIC として宣言できます。

compound-statement　BEGIN と END で囲まれ、セミコロンで区切られた SQL 文のセット。

tsql-compound-statement　Transact-SQL 文のバッチ。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

CREATE FUNCTION 文はデータベースに関数を作成します。所有者名を指定すれば、別のユーザーに対する関数を作成できます。パーミッションによる制御はありますが、関数は他の非集合関数とまったく同様に使用できます。

SQL SECURITY INVOKER が指定されている場合は、プロシージャーを呼び出すユーザーごとに注釈を行う必要があるためメモリ使用量が増えます。また、SQL SECURITY INVOKER が指定されている場合は、呼び出し側としても名前の決定が行われます。このため、すべてのオブジェクト名 (テーブル、プロシージャーなど) を該当する所有者で修飾する場合は注意が必要です。

NOT DETERMINISTIC と宣言されないかぎり、すべての関数は決定的として扱われます。決定的関数は、同じパラメーターに対して一貫した結果を返し、副次効果はありません。つまり、データベースサーバーは、同じパラメーターを持つ同じ関数が連続して 2 回呼び出されている場合は、どちらの呼び出しでも同じ結果が返され、クエリのセマンティックに不要な弊害は生じないものと見なします。

関数が結果セットを返す場合、出力パラメーターを設定したり戻り値を返したりすることはできません。

パーミッション

テンポラリ関数を作成するのでないかぎり、RESOURCE 権限が必要です。

Java 関数を含む外部関数には DBA 権限が必要です。

関連する動作

オートコミット、テンポラリ関数にも適用。

参照

- 「ALTER FUNCTION 文」421 ページ
- 「CREATE FUNCTION 文 [外部呼び出し]」550 ページ
- 「CREATE FUNCTION 文 [Web サービス]」555 ページ
- 「BEGIN 文」497 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「DROP FUNCTION 文」714 ページ
- 「RETURN 文」882 ページ
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』
- 「SQL データ型」85 ページ
- 「Transact-SQL のバッチ」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　CREATE FUNCTION は SQL/2008 標準のコア機能ですが、SQL Anywhere でサポートされている一部のコンポーネントはオプションの SQL 言語機能です。これらの機能のサブセットを次に示します。

  ○SQL SECURITY 句は、オプションの言語機能 T324 です。

○SQL 関数に LONG VARCHAR、LONG NVARCHAR、または LONG BINARY の値を渡す機能は、言語機能 T041 です。

○CREATE TABLE または DROP TRIGGER などの文を使用して SQL 関数内でスキーマオブジェクトを作成または変更する機能は、言語機能 T651 です。

○EXECUTE IMMEDIATE、PREPARE、および DESCRIBE などの文を使用して SQL 関数内で動的 SQL 文を使用する機能は、言語機能 T652 です。

CREATE FUNCTION 文のいくつかの句は、ベンダー拡張です。これらを以下に示します。

○TEMPORARY 句。

○ON EXCEPTION RESUME 句。

○特定のルーチンパラメーター用のオプションの DEFAULT 句。

○AS 句を使用した Transact-SQL 関数の指定。

○オプションの OR REPLACE 句。

- **Transact-SQL**　CREATE FUNCTION は、Adaptive Server Enterprise でサポートされています。Adaptive Server Enterprise では、関数パラメーターにオプションの IN キーワードはサポートされていません。

例

次の関数は、firstname 文字列と lastname 文字列を連結します。

```
CREATE FUNCTION fullname(
  firstname CHAR(30),
  lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
  DECLARE name CHAR(61);
  SET name = firstname || ' ' || lastname;
  RETURN (name);
END;
```

次の例は、最初の例で作成された fullname 関数を置き換えます。この関数が置き換えられると、ローカル変数 name が削除されます。

```
CREATE OR REPLACE FUNCTION fullname(
  firstname CHAR(30),
  lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
  RETURN ( firstname || ' ' || lastname );
END;
```

次の例は、fullname 関数の使用法を示します。

2 つの提供された文字列からフルネームを戻します。

```
SELECT fullname ( 'joe', 'smith' );
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

fullname('joe', 'smith')
joe smith

全従業員の名前をリストします。

```
SELECT fullname ( GivenName, Surname )
FROM Employees;
```

fullname (GivenName, Surname)
Fran Whitney
Matthew Cobb
Philip Chin
Julie Jordan
...

次の文は、Transact-SQL 構文を使用します。

```
CREATE FUNCTION DoubleIt( @Input INT )
RETURNS INT
AS
BEGIN
  DECLARE @Result INT
  SELECT @Result = @Input * 2
  RETURN @Result
END;
```

文 SELECT DoubleIt(5) は、10 の値を返します。

CREATE INDEX 文

指定したテーブルにインデックスを作成するか、マテリアライズドビューを作成します。

構文 1 - テーブルにインデックスを作成する

CREATE [**VIRTUAL**] [**UNIQUE**] [**CLUSTERED**] **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**]

構文 2 - マテリアライズドビューにインデックスを作成する

CREATE [**VIRTUAL**] [**UNIQUE**] [**CLUSTERED**] **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 ]
```

パラメーター

VIRTUAL 句　VIRTUAL キーワードは主に、Index Consultant によって使用されます。実行プランが Index Consultant によって評価されている間と、PLAN 関数が使用されている場合に、仮想インデックスは実際の物理的なインデックスのプロパティを模倣します。仮想インデックスを PLAN 関数と一緒に使用すると、実際のインデックスを作成するという時間とリソースを消費する作業をせずに、インデックスのパフォーマンスへの影響を調べることができます。

仮想インデックスは、他の接続からは参照できず、その接続が閉じたときに削除されます。仮想インデックスは、クエリの実際の実行プランを評価しているときは使用されないため、パフォーマンスの妨げにはなりません。

仮想インデックスは、最大 4 カラムまでを扱えます。

UNIQUE 句　UNIQUE 属性によって、インデックス内のすべてのカラムで同じ値を持つローがテーブルまたはマテリアライズドビュー内に複数存在しないようにします。UNIQUE を指定して、WITH NULLS NOT DISTINCT を指定しない場合は、各インデックスキーがユニークであるか、少なくとも 1 つのカラムで NULL を持つ必要があります。たとえば、2 つのエントリ ('a', NULL) と ('a', NULL) は、それぞれユニークと見なされます。

UNIQUE...WITH NULLS NOT DISTINCT を指定する場合は、インデックスキーが NULL 値に関係なくユニークである必要があります。たとえば、2 つのエントリ ('a', NULL) と ('a', NULL) は、同等でありユニークとは見なされません。

一意性制約とユニークインデックスには違いがあります。ユニークインデックスのカラムは NULL を使用できますが、一意性制約のカラムには使用できません。外部キーは、プライマリキーまたは一意性制約は参照できますが、ユニークインデックスは参照できません。ユニークインデックスは NULL の複数のインスタンスを含むことがあるからです。

プライマリキーや、一意性制約のカラムには FLOAT や DOUBLE などの概数値データ型を使用しないことをおすすめします。概数値データ型は、算術演算後の丸め誤差がでます。

ユニークインデックスには空間カラムは設定できません。

CLUSTERED 句　CLUSTERED 属性を使用すると、ローはインデックスのキーの順序で格納されます。データベースサーバーはキーの順序を保持しようとしますが、完全なクラスタは保証されません。

クラスタードインデックスが存在すると、LOAD TABLE 文はインデックスキーの順序でローを挿入し、INSERT 文はキーの順序の定義に従って、隣接するローがあるページに新しいローを配置しようとします。

IF NOT EXISTS 句　IF NOT EXISTS 属性を指定し、指定したインデックスがすでに存在する場合、変更は行われず、エラーは返されません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ASC | DESC 句　降順 (DESC) が明示的に指定されている場合以外、カラムは昇 (増加) 順で格納されます。昇順と降順 ORDER BY の両方に対してインデックスを使います。これはインデックスが昇順であっても降順であっても同様です。しかし、昇順と降順属性を混在させて ORDER BY を実行する場合、インデックスを使うのは同じ昇順と降順属性を使ってインデックスが作成されたときだけです。

function-name　function-name 句は、関数でインデックスを作成します。この句は、宣言済みのテンポラリテーブルまたはマテリアライズドビューでは使用できません。

この形式の CREATE INDEX 文は、次の操作を実行する際に便利です。

1. 計算カラム *column-name* をテーブルに追加します。カラムは、指定された関数である COMPUTE 句と指定された引数を使用して定義されます。指定できる関数の種類に関する制限については、CREATE TABLE 文の COMPUTE 句の説明を参照してください。カラムのデータ型は、関数の結果タイプに基づきます。

2. テーブルの既存のローに対して計算カラムを移植します。

3. カラムでインデックスを作成します。

   インデックスを削除しても、対応する計算カラムは削除されません。

IN | ON 句　デフォルトで、インデックスはそのテーブルまたはマテリアライズドビューと同じデータベースファイルの中に置かれます。インデックスを入れる DB 領域名を指定して、別のデータベースファイルにインデックスを入れることができます。この機能が便利なのは、主に大規模なデータベースがファイルサイズの制限を回避する場合、または複数のディスクデバイスを使用してパフォーマンスの改善が望める場合です。

新しいインデックスで物理インデックスを既存の論理インデックスと共有できる場合、IN 句は無視されます。

WITH NULLS NOT DISTINCT 句　この句はインデックスを UNIQUE と宣言した場合にのみ指定でき、インデックスキーで NULL がユニークでないことを指定できます。詳細については、UNIQUE 句を参照してください。

FOR OLAP WORKLOAD 句　FOR OLAP WORKLOAD を指定すると、データベースサーバーは特定の最適化を実行し、キーに関する統計情報を収集して、OLAP の負荷に関するパフォーマンスを向上させることができます。パフォーマンスの向上は、optimization_workload を OLAP に設定すると特に顕著になります。

備考

構文 1 はテーブルで使用し、構文 2 はマテリアライズドビューで使用します。

インデックスによってデータベースのパフォーマンスを改善できます。SQL Anywhere は物理インデックスと論理インデックスを使用します。物理インデックスは、ディスクに格納される実際のインデックス構造です。論理インデックスは、物理インデックスへの参照です。既存のインデックスに対する物理属性と同じインデックスを作成する場合、データベースサーバーは、既存の物理インデックスを共有する論理インデックスを作成します。通常、ユーザーが作成したインデックスは論理インデックスと見なされます。論理インデックスの実装に必要な場合、データ

ベースサーバーは物理インデックスを作成します。また、同じ物理インデックスを複数の論理インデックスで共有できます。

CREATEINDEX 文は、指定されたテーブルまたはマテリアライズドビューの特定のカラムにソートされたインデックスを作成できます。インデックスは、データベースに対して発行されたクエリのパフォーマンスを向上させたり、ORDER BY 句を使用してクエリをソートするときに自動的に使用されます。一度インデックスを作成すると、そのインデックスを検証する (VALIDATE INDEX) とき、変更する (ALTER INDEX) とき、削除する (DROPINDEX) とき以外は、再び参照されることはありません。

- **インデックスの所有者**　CREATE INDEX 文ではインデックスの所有者を指定できません。常にテーブルまたはマテリアライズドビューの所有者がインデックスの所有者となります。
- **ビュー上のインデックス**　マテリアライズドビュー上にはインデックスを作成できますが、通常のビューには作成できません。
- **インデックス名のスペース**　各インデックスには、特定のテーブルまたはマテリアライズドビューに対してユニークな名前を与えます。
- **排他的使用**　CREATE INDEX は、別の接続によって現在使用されているテーブルまたはマテリアライズドビューにこの文が影響する場合は常に阻止されます。CREATE INDEX は処理に時間がかかり、データベースサーバーは文が処理されている間は同じテーブルを参照する要求を処理しません。
- **自動的に作成されたインデックス**　SQL Anywhere はプライマリキー、外部キー、一意性制約のインデックスを自動的に作成します。これら自動作成インデックスは、テーブルと同じデータベースファイルに保持されます。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、この文は実行できません。

パーミッション

テーブルまたはマテリアライズドビューの所有者であるか、DBA 権限または REFERENCES パーミッションが必要です。

関連する動作

オートコミット。関数 (暗黙的な計算カラム) でインデックスを作成した場合にも、チェックポイントが発生します。

カラムの統計情報が更新されます (統計情報が存在しない場合は作成されます)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「DROP INDEX 文」715 ページ
- 「CREATE STATISTICS 文」643 ページ
- 「optimization_workload オプション」『SQL Anywhere サーバー データベース管理』
- 「高度：論理インデックスと物理インデックス」『SQL Anywhere サーバー SQL の使用法』
- 「インデックス」『SQL Anywhere サーバー SQL の使用法』
- 「クエリに対するインデックスコンサルタントの推奨内容の確認」『SQL Anywhere サーバー SQL の使用法』
- 「インデックスコンサルタント」『SQL Anywhere サーバー SQL の使用法』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』
- 「クラスタードインデックス」『SQL Anywhere サーバー SQL の使用法』
- 「計算カラム」『SQL Anywhere サーバー SQL の使用法』
- 「OLAP のサポート」『SQL Anywhere サーバー SQL の使用法』
- 「SQL Anywhere のサイズと数の制限」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

Employees テーブルで 2 カラムのインデックスを作成します。

```
CREATE INDEX employee_name_index
ON Employees
( Surname, GivenName );
```

ProductID カラム用に SalesOrderItems テーブル上でインデックスを作成します。

```
CREATE INDEX item_prod
ON SalesOrderItems
( ProductID );
```

SORTKEY 関数を使用して、Products テーブルの Description カラムにインデックスを作成し、Russian 照合に従ってソートします。関連動作として、この文は計算カラム desc_ru をテーブルに追加します。

```
CREATE INDEX ix_desc_ru
ON Products (
 SORTKEY( Description, 'rusdict' )
 AS desc_ru );
```

CREATE LOCAL TEMPORARY TABLE 文

プロシージャーが完了した後も、明示的に削除されるか接続が終了するまで保持されるローカルテンポラリテーブルをプロシージャー内で作成します。

構文

CREATE LOCAL TEMPORARY TABLE *table-name*
({ *column-definition* [*column-constraint* ...] | *table-constraint* | *pctfree* }, ... **)**
[**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS** | **NOT TRANSACTIONAL**]

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

パラメーター

ON COMMIT 句　デフォルトでは、テンポラリテーブルのローは COMMIT のときに削除されます。ON COMMIT 句を使用すると、COMMIT のときにローを保護できます。

NOT TRANSACTIONAL 句　状況によっては、NOT TRANSACTIONAL 句を使用するとパフォーマンスが向上します。これは、トランザクション単位でないテンポラリテーブルでの操作では、ロールバックログにエントリが作成されないためです。たとえば、テンポラリテーブルを使用するプロシージャーが COMMIT や ROLLBACK の介入を受けずに繰り返し呼び出される場合、NOT TRANSACTIONAL が有用です。

備考

プロシージャーの完了後も保持されるテーブルを作成する場合、DECLARE LOCAL TEMPORARY TABLE 文ではなく CREATE LOCAL TEMPORARY TABLE 文をプロシージャーに使用します。CREATE LOCAL TEMPORARY TABLE 文を使用して作成されたローカルテンポラリテーブルは、明示的に削除するか接続が終了するまで保持されます。

CREATE LOCAL TEMPORARY TABLE を使用して作成されたテーブルは、システムカタログの SYSTABLE ビューには表示されません。

CREATE LOCAL TEMPORARY TABLE を使用して IF 文で作成されたローカルテンポラリテーブルは、IF 文が完了した後も保持されます。

パーミッション

なし

関連する動作

なし

参照

- 「CREATE TABLE 文」651 ページ
- 「DECLARE LOCAL TEMPORARY TABLE 文」691 ページ
- 「複合文」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　CREATE LOCAL TEMPORARY TABLE は、SQL/2008 標準のオプションの言語機能 F531 の一部です。PCTFREE 句と NOT TRANSACTIONAL 句はベンダー拡張です。また、文によって定義されたカラムと制約の定義にも、ベンダー拡張の構文を含めることができます。SQL/2008 標準では、CREATE LOCAL TEMPORARY TABLE 文で作成されたテーブルがシステムカタログに表示されることが規定されていますが、SQL Anywhere ではそのようなことはありません。

- **Transact-SQL**　CREATE LOCAL TEMPORARY TABLE は、Adaptive Server Enterprise ではサポートされていません。Sybase Adaptive Server Enterprise では、特殊文字 # で始まるテーブル名を使用して CREATE TABLE 文でテンポラリテーブルを作成します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例は、TempTab というローカルテンポラリテーブルを作成します。

```
CREATE LOCAL TEMPORARY TABLE TempTab ( number INT )
ON COMMIT PRESERVE ROWS;
```

CREATE LOGIN POLICY 文

ログインポリシーを作成します。

構文

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** | *legal-option-value* }

パラメーター

policy-name ログインポリシーの名前。

policy-option-name ログインポリシーオプションの名前。オプションを指定しない場合は、ルートログインポリシーの値が適用されます。

policy-option-value ログインポリシーオプションに割り当てられている値。UNLIMITED を指定すると、制限は適用されません。

備考

ポリシーオプションを指定しない場合は、ルートログインポリシーからログインポリシーの値が取得されます。

すべての新しいデータベースには、ルートログインポリシーが含まれています。ルートログインポリシーの値を変更することはできますが、ポリシーは削除できません。次の表は、ルートログインポリシーのデフォルトオプションの概要を示します。

policy-option-name	説明	デフォルト値	適用対象
password_life_time	パスワードの変更が必要となるまでの最大日数。	Unlimited	すべてのユーザー (DBA 権限を持つユーザーを含む)。

policy-option-name	説明	デフォルト値	適用対象
password_grace_time	パスワードの有効期限が切れるまでの日数。この期間中は、ログインは許可されますが、デフォルトの post_login プロシージャーから警告が出ます。	0	すべてのユーザー (DBA 権限を持つユーザーを含む)。
password_expiry_on_next_login	このオプションの値が ON の場合、次回のログイン後にユーザーのパスワードは期限切れになります。	OFF	すべてのユーザー (DBA 権限を持つユーザーを含む)。
locked	このオプションの値が ON の場合、ユーザーは新しい接続を確立できません。DBA 権限を持つユーザーをロックすることはできません。sa_get_user_status システムプロシージャーの reason_locked カラムは、データベースサーバーで生成されたユーザーのロック理由を示す文字列を返します。	OFF	DBA 権限を持たないユーザー。
max_connections	ユーザーに許容される同時接続の最大数。	Unlimited	DBA 権限を持たないユーザー。
max_failed_login_attempts	前回のログイン成功以降、ユーザーがロックされるまでのログイン失敗の最大回数。	Unlimited	すべてのユーザー (DBA 権限を持つユーザーを含む)。 最後のログイン失敗から 1 分が経過すると、DBA ユーザーアカウントのロックは解除されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

policy-option-name	説明	デフォルト値	適用対象
max_days_since_login	同一ユーザーによる 2 回の連続するログインの間で許容される最大経過日数。	Unlimited	DBA 権限を持たないユーザー。
max_non_dba_connections	DBA 権限を持たないユーザーが確立できる同時接続の最大数。このオプションは、ルートログインポリシーでのみサポートされます。	Unlimited	DBA 権限を持たないユーザー。ルートログインポリシーのみ。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「ALTER LOGIN POLICY 文」424 ページ
- 「ALTER USER 文」482 ページ
- 「COMMENT 文」512 ページ
- 「CREATE USER 文」677 ページ
- 「DROP LOGIN POLICY 文」716 ページ
- 「DROP USER 文」735 ページ
- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』
- 「新しいログインポリシーの作成」『SQL Anywhere サーバー データベース管理』
- 「既存のユーザーへのログインポリシーの割り当て」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、Test1 ログインポリシーを作成します。この例では、無制限のパスワード有効期間が設定され、ユーザーは正しいパスワードの入力を 5 回まで試行できますが、これを超えるとアカウントがロックされます。

```
CREATE LOGIN POLICY Test1
password_life_time=UNLIMITED
max_failed_login_attempts=5;
```

CREATE MATERIALIZED VIEW 文

マテリアライズドビューを作成します。

構文

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* [,...] **)**

パラメーター

***alt-column-names* 句**　この句は、マテリアライズドビューでカラムの代替名を指定するときに使用します。代替カラム名を指定する場合、*alt-column-names* に含まれるカラム数が *select-statement* のカラム数に一致する必要があります。代替カラム名を指定しない場合、名前は *select-statement* の名前に設定されます。

IN 句　この句は、マテリアライズドビューを作成する DB 領域を指定するときに使用します。この句が指定されない場合、default_dbspace オプションで指定された DB 領域にマテリアライズドビューが作成されます。指定されている場合は、システム DB 領域が使用されます。

AS 句　この句は、マテリアライズドビューの移植に使用するデータを SELECT 文の形式で指定するときに使用します。マテリアライズドビュー定義は、ベーステーブルのみを参照できます。ビュー、他のマテリアライズドビュー、またはテンポラリテーブルは参照できません。*select-statement* にはカラム名を含めるか、エイリアス名を指定します。*alt-column-names* を指定する場合、これらの名前が *select-statement* に指定されたエイリアスの代わりに使用されます。

SELECT 文のカラム名は明示的に指定される必要があります。SELECT * の構成は使用できません。たとえば、CREATE MATERIALIZED VIEW matview AS SELECT * FROM table-name とは指定できません。また、*select-statement* ではオブジェクト名を完全に修飾してください。

CHECK 句　この句は、実際にビューを作成しないで文を検証するときに使用します。CHECK 句を指定すると、次のようになります。

○データベースサーバーは、CREATE MATERIALIZED VIEW がこの句なしで実行された場合に行う通常の言語チェックを実行し、通常どおり生成されたエラーが返されます。

○データベースサーバーは、実際のビューの作成を行いません。つまり、作成時に生じる恐れのある特定のエラーが生成されることはありません。たとえば、指定されたビュー名がすでに存在することを示すエラーは生成されません。このため、CHECK 句を使用すると、ビューの命名で競合することなく、ビュー定義の目的の変更をテストできます。

○CHECK IMMEDIATE REFRESH が使用される場合、データベースサーバーは即時ビューで構文が有効であるかを検証し、エラーを返します。

○データベースへの変更は行われず、トランザクションログには何も記録されません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○文実行の開始時に暗黙的なコミットがあり、実行中に取得されたすべてのロックを解放するロールバックが最後に行われます。

参照

- 「マテリアライズドビューの制限」『SQL Anywhere サーバー SQL の使用法』
- 「追加の DB 領域の考慮事項」『SQL Anywhere サーバー データベース管理』

備考

マテリアライズドビューを作成すると、これは手動ビューとなり、初期化されていない状態です。つまり、再表示タイプは手動であり、ビューはリフレッシュされていません (データは移植されています)。このビューを初期化するには、REFRESH MATERIALIZED VIEW 文を実行するか、sa_refresh_materialized_views システムプロシージャーを使用します。

マテリアライズドビューを暗号化し、PCTFREE 設定を変更し、再表示タイプを変更し、オプティマイザーによって使用を有効または無効にできます。ただし、マテリアライズドビューを作成してから、ALTER MATERIALIZED VIEW を使用してこれらの設定を変更します。マテリアライズドビュー作成時のデフォルト値は、次のとおりです。

●NOT ENCRYPTED

●ENABLE USE IN OPTIMIZATION

●PCTFREE は、データベースのページサイズに応じて、ページサイズが 4 KB の場合は 200 バイト、ページサイズが 2 KB の場合は 100 バイトに設定されます。

●MANUAL REFRESH

マテリアライズドビューを作成するには、いくつかのデータベースとサーバーのオプションが有効になっている必要があります。

sa_recompile_views システムプロシージャーは、マテリアライズドビューに影響しません。

参照

- 「マテリアライズドビューの制限」『SQL Anywhere サーバー SQL の使用法』
- 「REFRESH MATERIALIZED VIEW 文」867 ページ

パーミッション

マテリアライズドビュー定義内のテーブルに対する RESOURCE 権限と SELECT パーミッションが必要です。別のユーザーのマテリアライズドビューを作成するには、DBA 権限が必要です。

関連する動作

実行中に、CREATE MATERIALIZED VIEW 文は、マテリアライズドビューから参照されるすべてのテーブルに排他ロックをかけますが、ブロックは実行しません。参照テーブルのいずれかがロックできない場合、文は失敗し、エラーが返されます。

参照

- 「マテリアライズドビュー」『SQL Anywhere サーバー SQL の使用法』
- 「高度：マテリアライズドビューのステータスとプロパティ」『SQL Anywhere サーバー SQL の使用法』
- 「ALTER MATERIALIZED VIEW 文」427 ページ
- 「DROP MATERIALIZED VIEW 文」717 ページ
- 「REFRESH MATERIALIZED VIEW 文」867 ページ
- 「CREATE VIEW 文」681 ページ
- 「sa_refresh_materialized_views システムプロシージャー」1158 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、SQL Anywhere サンプルデータベースに格納されている従業員に関する機密情報を含むマテリアライズドビューを作成します。この後、ビューを初期化して使用するには、例に示すように REFRESH MATERIALIZED VIEW 文を実行する必要があります。

```
CREATE MATERIALIZED VIEW EmployeeConfid2 AS
SELECT EmployeeID, Employees.DepartmentID,
   SocialSecurityNumber, Salary, ManagerID,
   Departments.DepartmentName, Departments.DepartmentHeadID
FROM Employees, Departments
WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid2;
```

CREATE MESSAGE 文 [T-SQL]

PRINT 文と RAISERROR 文によって使用される、ユーザー定義メッセージをシステムテーブル ISYSUSERMESSAGE に追加します。

構文

CREATE MESSAGE *message-number* **AS** *message-text*

message-number : *integer*

message-text : *string*

パラメーター

message-number　追加するメッセージのメッセージ番号。ユーザー定義メッセージのメッセージ番号は 20000 以上にしてください。

message-text　追加するメッセージのテキスト。最大長は、255 バイトです。PRINT と RAISERROR は、メッセージテキストのプレースホルダーを認識します。1 つのメッセージには、ユニークなプレースホルダーを任意の順序で 20 個まで含めることができます。これらのプレースホルダーは、メッセージのテキストをクライアントに送信するときにメッセージの後に付けた任意の引数のフォーマット文字列に置換されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メッセージを文法的構文が異なる言語に翻訳するときに順番を変えられるように、プレースホルダーには番号が付けられます。引数のプレースホルダーの形式は "%nn!" です。パーセント記号 (%)、nn (1 ～ 20 の整数)、感嘆符 (!) の順に表されます。整数は引数の引数リスト内の位置を表します。"%1!" が最初の引数で、2 番目の引数は "%2!"、のようになります。

sp_addmessage の *language* 引数に対応するパラメーターはありません。

備考

CREATE MESSAGE はメッセージ番号をメッセージ文字列に関連付けます。このメッセージ番号は PRINT 文と RAISERROR 文で使用できます。

パーミッション

RESOURCE 権限が必要です。

関連する動作

オートコミット。

参照

- 「PRINT 文 [T-SQL]」859 ページ
- 「RAISERROR 文」862 ページ
- 「DROP MESSAGE 文」718 ページ
- 「SYSUSERMESSAGE システムビュー」1329 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　CREATE MESSAGE には、Adaptive Server Enterprise の sp_addmessage システムプロシージャーで提供される機能が備えられています。

CREATE MIRROR SERVER 文

> **注意**
> 読み込み専用のスケールアウトとデータベースミラーリングには、それぞれ別のライセンスが必要です。「別途ライセンスが必要なコンポーネント」『SQL Anywhere 12 紹介』を参照してください。

データベースミラーリングまたは読み込み専用のスケールアウトに使用されているミラーサーバーを作成するか、置き換えます。

構文

CREATE [**OR REPLACE**] **MIRROR SERVER** *mirror-server-name*
AS { **PRIMARY** | **MIRROR** | **ARBITER** | **PARTNER** | **COPY** }
[{ **FROM SERVER** *parent-name* [**OR SERVER** *server-name*] | **USING AUTO PARENT** }]
[*server-option* = *string* [...]]

parent-name :
server-name | **PRIMARY**

server-option :
connection_string
logfile
preferred
state_file

パラメーター

- **OR REPLACE 句**　CREATE MIRROR SERVER は、ミラーサーバーを作成します。指定された名前のミラーサーバーがデータベースにすでに存在する場合は、エラーが返されます。

  OR REPLACE を指定すると、ミラーサーバーがデータベースに存在しない場合は作成され、存在する場合は置き換えられます。使用中のミラーサーバーを置き換えようとすると、エラーが返されます。

- **AS 句**　次のいずれかのサーバータイプを指定できます。

  - **PRIMARY**　PRIMARY タイプのミラーサーバーは、実際のデータベースサーバーではなく、仮想サーバーまたは論理サーバーを定義します。このサーバーの名前は、データベースの代替サーバー名です。アプリケーションで代替サーバー名を使用して、現在プライマリサーバーとして動作しているサーバーに接続できます。PRIMARY としてマーク付けされているサーバーも、ミラーサーバーで使用されている接続文字列を定義して、現在プライマリサーバーとして動作しているサーバーに接続します。また、スケールアウトシステムで新しいコピーノードをルートサーバーに接続する方法を定義します。データベースで使用できる PRIMARY サーバーは 1 つのみです。

  - **MIRROR**　MIRROR タイプのミラーサーバーは、実際のデータベースサーバーではなく、仮想サーバーまたは論理サーバーを定義します。このサーバーの名前は、データベースの代替ミラーサーバー名です。アプリケーションで代替ミラーサーバー名を使用して、現在読み込み専用ミラーとして動作しているサーバーに接続できます。データベースで使用できる MIRROR サーバーは 1 つのみです。

  - **ARBITER**　データベースミラーリングシステムでは、監視サーバーは、いずれの PARTNER サーバーがデータベースの所有権を取得するかを決定するのに役立ちます。監視サーバーは、パートナーサーバーが監視サーバーへの接続に使用できる接続文字列を使用して定義します。データベースで使用できる ARBITER サーバーは 1 つのみです。

  - **PARTNER**　データベースミラーリングシステムでは、PARTNER として定義されたサーバーは、プライマリサーバーとなってデータベースの所有権を取得できます。データベースミラーリングを行うには、2 つの PARTNER サーバーを定義する必要があり、両方のサーバーに接続文字列とステータスファイルが必要です。ミラーサーバー名は、-n サーバーオプションで指定されたデータベースサーバー名に対応している必要があり、connection_string ミラーサーバーオプションで指定された SERVER 接続文字列と一致している必要があります。

    読み込み専用のスケールアウトシステムでは、1 つの PARTNER サーバーを定義します。このサーバーはルートサーバーであり、読み込みと書き込みの両方の処理が許可されたデータベースの唯一のコピーを実行します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **COPY**　読み込み専用のスケールアウトシステムでは、この値は、データベースサーバーがコピーノードであることを指定します。このサーバー上のデータベース接続はすべて、読み込み専用です。ミラーサーバー名は、-n サーバーオプションで指定されたデータベースサーバー名に対応している必要があり、connection_string ミラーサーバーオプションで指定された SERVER 接続文字列と一致している必要があります。スケールアウトシステムにコピーノードを明示的に定義する必要はありません。接続時に、ルートノードによってコピーノードを定義できます。

● **FROM SERVER 句**　この句は、COPY タイプのミラーサーバーに対してのみ指定できます。この句は、ミラーリングシステムまたはスケールアウトシステムのサーバーのツリーを構成し、コピーノードがどのサーバーからトランザクションログページを取得するかを示します。

ミラーサーバー名または PRIMARY を使用して、親を指定できます。コピーノードの代替の親は、OR SERVER 句を使用して指定できます。

2 レベル (パートナーノードとコピーノード) のみが存在するデータベースミラーリングシステムでは、コピーノードが、現在のプライマリサーバーまたはミラーサーバーからトランザクションログページを取得します。

コピーノードは、データベースに格納されているコピーノードのミラーサーバー定義を使用して、接続先となるサーバーを判断します。コピーノードの定義から親の定義を探すことができ、親の定義から親に接続するための接続文字列を取得できます。

スケールアウトシステムにコピーノードを明示的に定義する必要はありません。接続時に、ルートノードによってコピーノードを定義できます。

● **OR SERVER 句**　OR SERVER 句を使用して、コピーノードの代替の親を指定します。

● **USING AUTO PARENT 句**　この句を指定すると、このサーバーの親がプライマリサーバーによって割り当てられます。

● ***server-option* 句**　次のオプションがサポートされています。

○ **connection_string サーバーオプション**　サーバーに接続するための接続文字列を指定します。ユーザー ID とパスワードは必要ありません。1 つのミラーサーバーを別のミラーサーバーに接続するときにユーザー ID とパスワードは使用されないため、ミラーサーバーの接続文字列にはユーザー ID とパスワードを含めないでください。

接続パラメーターのリストについては、「接続パラメーター」『SQL Anywhere サーバーデータベース管理』を参照してください。

○ **logfile サーバーオプション**　データベースミラーリングが使用されている場合に、ミラーサーバー間で送信されるファイル (1 行に 1 つの要求を含む) のロケーションを指定します。このファイルは、デバッグ専用です。

○ **preferred サーバーオプション**　サーバーがミラーリングシステムにおいて優先サーバーであるかどうかを指定します。YES または NO のいずれかを指定できます。優先サーバーは、可能な場合は常にプライマリサーバーとしての役割を果たします。このオプションは、PARTNER サーバーを定義する場合に指定します。

○ **state_file サーバーオプション**　ミラーリングシステムに関するステータス情報の管理に使用されるファイルのロケーションを指定します。データベースミラーリングには、このオプションは必須です。PARTNER タイプのサーバーには、ステータスファイルを指定してください。監視サーバーの場合は、サーバーの起動コマンドの一部としてロケーションが指定されます。

備考

データベースミラーリングシステムでは、ミラーサーバーのタイプに PRIMARY、MIRROR、ARBITER、または PARTNER を使用できます。

読み込み専用のスケールアウトシステムでは、ミラーサーバーのタイプに PRIMARY、PARTNER、または COPY を使用できます。

PARTNER、ARBITER、または COPY タイプのサーバーのミラーサーバー名は、ミラーリングシステムの一部であるデータベースサーバーの名前 (-n サーバーオプションで使用される名前) と一致している必要があります。これにより、各データベースサーバーで、サーバー自体の定義とその親の定義を見つけることができるようになります。また、すべてのコピーノードサーバーの名前がユニークになるように指定します。

データベースミラーリングシステムで、コピーしているデータベースの監視サーバーとしてコピーノードを使用するには、高可用性システムのデータベースサーバーのいずれのサーバー名とも一致しない名前で監視サーバーを作成します。このように設定すると、ミラーサーバー定義において、監視サーバーの名前が監視サーバーの接続文字列が保持するプレースホルダーとして使用されます。

パーミッション

DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「SYSMIRRORSERVER システムビュー」1280 ページ
- 「データベースミラーリング」『SQL Anywhere サーバー データベース管理』
- 「SQL Anywhere の読み込み専用のスケールアウト」『SQL Anywhere サーバー データベース管理』
- 「SET MIRROR OPTION 文」908 ページ
- 「ALTER MIRROR SERVER 文」430 ページ
- 「COMMENT 文」512 ページ
- 「DROP MIRROR SERVER 文」719 ページ
- 「子コピーノードを追加する方法」『SQL Anywhere サーバー データベース管理』
- 「ステータス情報ファイル」『SQL Anywhere サーバー データベース管理』
- 「優先データベースサーバー」『SQL Anywhere サーバー データベース管理』
- 「コピーノードへの親の自動割り当て」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、データベースミラーリングシステムでプライマリサーバーとして使用できるミラーサーバーを作成します。

```
CREATE MIRROR SERVER "scaleout_primary"
    AS PRIMARY
    connection_string = 'server=scaleout_primary;host=winxp-2:6871,winxp-3:6872';
```

次の文は、データベースミラーリングシステムでミラーサーバーとして使用できるミラーサーバーを作成します。

```
CREATE MIRROR SERVER "scaleout_mirror"
AS MIRROR
connection_string = 'server=scaleout_mirror;host=winxp-2:6871,winxp-3:6872';
```

次の文は、データベースミラーリングシステムで監視サーバーとして使用できるミラーサーバーを作成します。

```
CREATE MIRROR SERVER "scaleout_arbiter"
AS ARBITER
connection_string = 'server=scaleout_arbiter;host=winxp-4:6870';
```

次の文は、データベースミラーリングシステムでパートナーサーバーとして使用できるミラーサーバーを作成します。

```
CREATE MIRROR SERVER "scaleout_server1"
AS PARTNER
connection_string = 'server=scaleout_server1;HOST=winxp-2:6871'
state_file = 'c:¥server1¥server1.state';
```

CREATE PROCEDURE 文 [外部呼び出し]

ネイティブプロシージャーまたは外部プロシージャーへのインターフェイスを作成します。

構文

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** }

parameter :
[*parameter-mode*] *parameter-name data-type* [**DEFAULT** *expression*]
| **SQLCODE**
| **SQLSTATE**

parameter-mode :
IN

| **OUT**
| **INOUT**

native-call :
[*operating-system*:]*function-name*@*library*

result-column :
column-name data-type

c-call :
[*operating-system*:]*function-name*@*library*; ...

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 method-signature*

operating-system :
Unix

method-signature :
([*field-descriptor*, ...]) *return-descriptor*

field-descriptor と *return-descriptor* :
{ **Z**
| **B**
| **S**
| **I**
| **J**
| **F**
| **D**
| **C**
| **V**
| [*descriptor*
| L*class-name*;
}

パラメーター

外部プロシージャーまたはネイティブプロシージャーを呼び出す永続的なストアドプロシージャーの作成には、さまざまなプログラミング言語を使用できます。PROC は PROCEDURE の同義語として使用できます。

OR REPLACE 句　OR REPLACE を指定すると、新しいプロシージャーが作成されるか、同じ名前の既存のプロシージャーが置き換えられます。この句によって、プロシージャーの定義は変更されますが、既存のパーミッションは保持されます。使用中のプロシージャーを置き換えようとすると、エラーが返されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター　パラメーター名は、カラム名など他のデータベース識別子に対するルールに従って付けてください。これらは有効な SQL データ型にする必要があります。

パラメーターには、IN、OUT、INOUT のいずれかのキーワードをプレフィクスとして付けることができます。これらの値のいずれも指定しない場合、パラメーターはデフォルトで INOUT になります。キーワードには次の意味があります。

○ **IN**　このパラメーターは、プロシージャーに値を与える式です。

○ **OUT**　このパラメーターは、プロシージャーから値を受け取ることがある変数です。

○ **INOUT**　このパラメーターはプロシージャーに値を与え、プロシージャーから新しい値を受け取ることがある変数です。

CALL 文を使ってプロシージャーを実行する場合、必ずしもすべてのパラメーターを指定する必要はありません。CREATE PROCEDURE 文の中にデフォルト値がある場合、不明のパラメーターにデフォルト値を割り当てます。CALL に引数が指定されておらず、デフォルトも設定されていない場合には、エラーが発生します。

SQLSTATE と SQLCODE は、プロシージャーが終了するときに、SQLSTATE または SQLCODE 値を出力する、特別な OUT パラメーターです。SQLSTATE と SQLCODE の特別値は、プロシージャーのリターンステータスのテストを目的として、プロシージャー呼び出しの直後にチェックできます。

SQLSTATE と SQLCODE 特別値は、その次の SQL 文によって修正されます。SQLSTATE と SQLCODE をプロシージャー引数として与えると、リターンコードは変数の中に格納されます。

OR REPLACE (CREATE OR REPLACE PROCEDURE) を指定すると、新しいプロシージャーが作成されるか、同じ名前の既存のプロシージャーが置き換えられます。この句によって、プロシージャーの定義は変更されますが、既存のパーミッションは保持されます。使用中のプロシージャーを置き換えようとすると、エラーが返されます。

TEMPORARY 外部呼び出しプロシージャーは作成できません。

RESULT 句　RESULT 句は結果セットのカラムの数と型を宣言します。RESULT キーワードに続くカッコで囲まれたリストは、結果カラムの名前と型を定義します。CALL 文が記述されていると、この情報を Embedded SQL DESCRIBE または ODBC SQLDescribeCol が返します。

Embedded SQL (LANGUAGE C_ESQL32、LANGUAGE C_ESQL64) 外部プロシージャーまたは ODBC (LANGUAGE C_ODBC32、LANGUAGE C_ODBC64) 外部プロシージャーは、結果セットを返さないか、1 つの結果セットを返すことができます。

Perl または PHP (LANGUAGE PERL、LANGUAGE PHP) 外部プロシージャーは、結果セットを返すことができません。データベースサーバーによってロードされるネイティブ関数を呼び出すプロシージャーも、結果セットを返すことができません。

CLR または Java (LANGUAGE CLR、LANGUAGE JAVA) 外部プロシージャーは、結果セットを返さないか、1 つ以上の結果セットを返すことができます。

プロシージャーは、その実行方法に応じて、それぞれカラム数が異なる複数の結果セットを生成する場合があります。たとえば次のプロシージャーは、2 カラムを返す場合も、1 カラムを返す場合もあります。

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
  IF formal = 'n' THEN
    SELECT GivenName
    FROM Employees
  ELSE
    SELECT Surname, GivenName
    FROM Employees
  END IF
END;
```

これらの結果セットプロシージャーは RESULT 句を指定しないで記述するか、Transact-SQL で記述します。これらの使用には、次の制約があります。

○ **Embedded SQL**　正しい形式の結果セットを取得するには、結果セットのカーソルが開かれてからローが返されるまでの間に、プロシージャーコールを記述 (DESCRIBE) します。DESCRIBE 文の CURSOR *cursor-name* 句は必須です。

○ **ODBC、OLE DB、ADO.NET**　変数結果セットプロシージャーは、これらのインターフェイスを使用するアプリケーションで使用できます。結果セットの記述は、ドライバーまたはプロバイダーによって実行されます。

○ **Open Client アプリケーション**　変数結果セットプロシージャーは Open Client アプリケーションで使用できます。

プロシージャーが結果セットを 1 つしか返さない場合、RESULT 句を使用してください。この句を使用すると、カーソルがオープンした後で ODBC と Open Client のアプリケーションが結果セットを記述し直すのを防ぐことができます。

複数の結果セットを処理するために ODBC は、プロシージャーが定義した結果セットではなく、現在実行中のカーソルを記述します。したがって、ODBC はいつもプロシージャー定義の RESULT 句内で定義されているカラム名を記述するわけではありません。この問題を回避するには、結果セットを生成する SELECT 文でカラムエイリアスを使用します。

NO RESULT SET 句　このプロシージャーによって結果セットが返されないことを宣言します。この宣言によって、パフォーマンスが向上することがあります。

DYNAMIC RESULT SETS 句　この句は、LANGUAGE CLR 呼び出しまたは LANGUAGE JAVA 呼び出しで使用します。DYNAMIC RESULT SETS 句を使用して、プロシージャーによって返される動的結果セットの数を指定します。RESULT 句が指定され、DYNAMIC RESULT SETS 句が指定されていない場合、動的結果セットの数は 1 とみなされます。RESULT 句も DYNAMIC RESULT SETS 句も指定されていない場合、結果セットは予測されず、結果セットが生成されるとエラーになります。

C_ESQL32、C_ESQL64、C_ODBC32、C_ODBC64 外部環境も結果セット (CLR や JAVA など) を返しますが、これらは 1 つの動的結果セットに制限されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Perl または PHP (LANGUAGE PERL、LANGUAGE PHP) 外部関数を呼び出すプロシージャーは、結果セットを返すことができません。データベースサーバーによってロードされるネイティブ関数を呼び出すプロシージャーも、結果セットを返すことができません。

SQL SECURITY 句　SQL SECURITY 句は、INVOKER (プロシージャーを呼び出すユーザー) または DEFINER (プロシージャーを所有するユーザー) としてプロシージャーが実行されるかどうかを定義します。デフォルトは DEFINER です。外部呼び出しの場合、この句は、外部環境での修飾されていないオブジェクト参照に対して所有者のコンテキストを確立します。

SQL SECURITY INVOKER が指定されている場合は、プロシージャーを呼び出すユーザーごとに注釈を行う必要があるためメモリ使用量が増えます。また、SQL SECURITY INVOKER が指定されている場合は、呼び出し側としても名前の決定が行われます。このため、すべてのオブジェクト名 (テーブル、プロシージャーなど) を該当する所有者で修飾する場合は注意が必要です。たとえば、user1 が次のプロシージャーを作成するとします。

```
CREATE PROCEDURE user1.myProcedure()
  RESULT( columnA INT )
  SQL SECURITY INVOKER
  BEGIN
    SELECT columnA FROM table1;
  END;
```

user2 がこのプロシージャーを実行しようとし、テーブル user2.table1 が存在**しない**場合、テーブルルックアップエラーが生じます。さらに、user2.table1 が存在**する**場合は、意図する user1.table1 の代わりにこのテーブルが使用されます。このような状況を防ぐには、文においてテーブル参照を修飾します (単なる table1 ではなく、user1.table1 とします)。

EXTERNAL NAME 句　LANGUAGE 属性なしで EXTERNAL NAME 句を使用するプロシージャーは、C などのプログラミング言語で記述されたネイティブ関数へのインターフェイスを定義します。ネイティブ関数は、データベースサーバーによってそのアドレス領域にロードされます。

library 名にはファイル拡張子を付けることができます。これは通常、Windows では *.dll*、UNIX では *.so* です。拡張子がない場合、ライブラリに対するプラットフォーム固有のデフォルトのファイル拡張子が追加されます。次に、正式な例を示します。

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
```

上記の EXTERNAL NAME 句を、プラットフォーム固有のデフォルト値を使用して簡単に記述すると、次のようになります。

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mylib';
```

呼び出されると、関数を含むライブラリがデータベースサーバーのアドレス領域にロードされます。ネイティブ関数は、サーバーの一部として実行されます。この場合、関数が原因で障害が発生すると、データベースサーバーは終了されます。したがって、外部環境での関数のロードと実行には、LANGUAGE 属性を使用することをおすすめします。関数が原因で外部環境に障害が発生しても、データベースサーバーは実行し続けます。

operating-system がサポートされている構文で *operating-system* を指定しないと、プロシージャーがすべてのプラットフォームで稼働すると見なされます。いずれかの呼び出しで **Unix** を指定すると、他の呼び出しは Windows 用であると見なされます。

- **EXTERNAL NAME '*c-call*' LANGUAGE {C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 } 句**　コンパイル済みネイティブ C 関数をデータベースサーバー内でなく外部環境で呼び出すには、ストアドプロシージャーまたはファンクションを EXTERNAL NAME 句で定義し、後続の LANGUAGE 属性で C_ESQL32、C_ESQL64、C_ODBC32、C_ODBC64 のいずれか 1 つを指定します。

  LANGUAGE 属性が指定されると、その関数を含むライブラリが外部プロセスによってロードされ、外部関数がその外部プロセスの一部として実行されます。この場合、関数が原因で障害が発生しても、データベースサーバーは実行し続けます。

  次に、プロシージャー定義の例を示します。

```
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* LANGUAGE CLR 句**　.NET 関数を外部環境で呼び出すには、プロシージャーインターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE CLR 属性を指定します。

  CLR ストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファンクションのコードは C# または Visual Basic などの .NET 言語で記述され、その実行はデータベースサーバーの外側 (つまり別の .NET 実行ファイル内) で行われます。

  次に、プロシージャー定義の例を示します。

```
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* LANGUAGE PERL 句**　Perl 関数を外部環境で呼び出すには、プロシージャーインターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE PERL 属性を指定します。

  Perl ストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファンクションのコードは Perl で記述され、その実行はデータベースサーバーの外側 (つまり Perl 実行インスタンス内) で行われます。

  次に、プロシージャー定義の例を示します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
CREATE PROCEDURE PerlWriteToConsole( IN str LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME '<file=PerlConsoleExample>
   WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```

- **EXTERNAL NAME *php-call* LANGUAGE PHP 句**　PHP 関数を外部環境で呼び出すには、プロシージャーインターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE PHP 属性を指定します。

  PHP ストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファンクションのコードは PHP で記述され、その実行はデータベースサーバーの外側 (つまり PHP 実行インスタンス内) で行われます。

  次に、プロシージャー定義の例を示します。

```
CREATE PROCEDURE PHPPopulateTable()
NO RESULT SET
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;
```

- **EXTERNAL NAME *java-call* LANGUAGE JAVA 句**　Java メソッドを外部環境で呼び出すには、プロシージャーインターフェイスを EXTERNAL NAME 句で定義し、それに続いて LANGUAGE JAVA 属性を指定します。

  Java とのインターフェイスとなるストアドプロシージャーまたはファンクションの動作は、SQL ストアドプロシージャーまたはファンクションと同じです。ただし、プロシージャーまたはファンクションのコードは Java で記述され、その実行はデータベースサーバーの外側 (つまり Java VM 内) で行われます。

  次に、プロシージャー定義の例を示します。

```
CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )
NO RESULT SET
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

備考

CREATE PROCEDURE 文はデータベースにプロシージャーを作成します。DBA 権限がある ユーザーは、所有者を指定することによって他のユーザーのプロシージャーを作成できます。プロシージャーは CALL 文で呼び出します。

ストアドプロシージャーが結果セットを返す場合、出力パラメーターを設定したり戻り値を返したりすることはできません。

複数のプロシージャーからテンポラリテーブルを参照する場合、テンポラリテーブル定義が矛盾していたり、テーブルを参照する文がキャッシュされていたりすると、問題が発生する可能性があります。

パーミッション

テンポラリプロシージャーを作成するのでないかぎり、RESOURCE 権限が必要です。

外部プロシージャーを作成する場合、および別のユーザーのプロシージャーを作成する場合、DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「プロシージャー内でのテンポラリテーブルの参照」『SQL Anywhere サーバー SQL の使用法』
- 「ALTER PROCEDURE 文」433 ページ
- 「CALL 文」503 ページ
- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [外部呼び出し]」550 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「CREATE PROCEDURE 文 [T-SQL]」599 ページ
- 「DROP PROCEDURE 文」720 ページ
- 「GRANT 文」782 ページ
- 「SQL データ型」85 ページ
- 「ESQL 外部環境と ODBC 外部環境」『SQL Anywhere サーバー プログラミング』
- 「SQL Anywhere 外部呼び出しインターフェイス」『SQL Anywhere サーバー プログラミング』
- 「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』
- 「PERL 外部環境」『SQL Anywhere サーバー プログラミング』
- 「CLR 外部環境」『SQL Anywhere サーバー プログラミング』
- 「PHP 外部環境」『SQL Anywhere サーバー プログラミング』
- 「Java 外部環境」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　外部言語環境の CREATE PROCEDURE は SQL/2008 標準のコア機能ですが、SQL Anywhere でサポートされている一部のコンポーネントはオプションの SQL/2008 言語機能です。これらの機能のサブセットを次に示します。

  ○SQL SECURITY 句は、オプションの SQL/2008 言語機能 T324 です。

  ○外部プロシージャーに LONG VARCHAR、LONG NVARCHAR、または LONG BINARY 値を渡す機能は、SQL/2008 言語機能 T041 です。

  ○CREATE TABLE または DROP TRIGGER などの文を使用して外部プロシージャー内でスキーマオブジェクトを作成または変更する機能は、SQL/2008 言語機能 T653 です。

  ○CONNECT、EXECUTE IMMEDIATE、PREPARE、および DESCRIBE などの文を使用して外部プロシージャー内で動的 SQL 文を使用する機能は、SQL/2008 言語機能 T654 です。

  ○JAVA 外部プロシージャーには、SQL/2008 言語機能 J621 が埋め込まれています。

  CREATE PROCEDURE 文のいくつかの句は、ベンダー拡張です。これらを以下に示します。

  ○LANGUAGES 句での C_ESQL32、C_ESQL64、C_ODBC32、C_ODBC64、CLR、PERL、および PHP のサポートは、ベンダー拡張です。SQL/2008 標準では、オプションの言語機能 B122 として "C" が *environment-name* としてサポートされています。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○*external-call* のフォーマットは実装依存です。

○RESULT 句と NO RESULT SET 句は、ベンダー拡張です。SQL/2008 標準では、RETURNS 句が使用されます。

○特定のルーチンパラメーターのオプションの DEFAULT 句は、ベンダー拡張です。

○オプションの OR REPLACE 句は、ベンダー拡張です。

- **Transact-SQL**　外部ルーチンに対する CREATE PROCEDURE は、Adaptive Server Enterprise でサポートされています。Adaptive Server Enterprise では、C 言語および Java 言語の外部ルーチンがサポートされています。

CREATE PROCEDURE 文 [Web サービス]

HTTP サーバーへの HTTP 要求または SOAP 要求を作成するユーザー定義の Web クライアントプロシージャーを作成します。

構文

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** }]

soap-type-spec-string :
SOAP[:{ **RPC** | **DOC** }

parameter :
parameter-mode parameter-name data-type [**DEFAULT** *expression*]

parameter-mode :
IN
| **OUT**
| **INOUT**

url-string :
{ **HTTP** | **HTTPS** | **HTTPS_FIPS** }:*//*[*user*:*password*@]*hostname*[:*port*][*/path*]

protocol-option-string
[*http-option-list*]
[, *soap-option-list*]

http-option-list :
HTTP(
[**CH[UNK]={ ON | OFF | AUTO }**]
[; **VER[SION]={ 1.0 | 1.1 }**]
)

soap-option-list :
SOAP(OP[ERATION]=*soap-operation-name***)**

パラメーター

Web サービスクライアントプロシージャーの作成または置き換えができます。PROC は PROCEDURE の同義語として使用できます。

SOAP 要求の場合、プロシージャー名はデフォルトで SOAP 操作名として使用されます。詳細については、SET 句を参照してください。

> **注意**
> TEMPORARY Web サービスプロシージャーを作成することはできません。

OR REPLACE 句　OR REPLACE を指定すると、新しいプロシージャーが作成されるか、同じ名前の既存のプロシージャーが置き換えられます。この句によって、プロシージャーの定義は変更されますが、既存のパーミッションは保持されます。使用中のプロシージャーを置き換えようとすると、エラーが返されます。

パラメーター　パラメーター名は、カラム名など他のデータベース識別子に対するルールに従って付けてください。これらは有効な SQL データ型にする必要があります。

FLOAT、INT などの型指定のあるデータの転送をサポートするのは SOAP 要求のみです。HTTP 要求は文字列の転送のみサポートしているため、CHAR 型に制限されます。

CALL 文を使ってプロシージャーを実行する場合、必ずしもすべてのパラメーターを指定する必要はありません。CREATE PROCEDURE 文の中にデフォルト値がある場合、不明のパラメーターにデフォルト値を割り当てます。CALL に引数が指定されておらず、デフォルトも設定されていない場合には、エラーが発生します。

パラメーターには、IN、OUT、INOUT のいずれかのキーワードをプレフィクスとして付けることができます。これらの値のいずれも指定しない場合、パラメーターはデフォルトで INOUT になります。キーワードには次の意味があります。

○ **IN**　このパラメーターは、プロシージャーに値を与える式です。

○ **OUT**　このパラメーターは、プロシージャーから値を受け取ることがある変数です。

○ **INOUT**　このパラメーターはプロシージャーに値を与え、プロシージャーから新しい値を受け取ることがある変数です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

RESULT 句　RESULT 句は SELECT 文でプロシージャーを使用するために必要です。RESULT 句は 2 つのカラムを返します。1 番目のカラムには HTTP 応答ヘッダー、ステータス、および応答本文属性が含まれ、2 番目のカラムにはこれらの属性の値が含まれます。RESULT 句では 2 文字のデータ型を指定します。たとえば、VARCHAR や LONG VARCHAR などです。RESULT 句が指定されていない場合、デフォルトのカラム名は Attribute および Value で、それらのデータ型は LONG VARCHAR です。

URL 句　Web サービスの URI を指定します。オプションのユーザー名とパスワードのパラメーターは、HTTP 基本認証に必要なクレデンシャルとして機能します。HTTP 基本認証は、ユーザーとパスワードの情報を base-64 でエンコードし、HTTP 要求の Authentication ヘッダーに渡します。この方法で指定すると、ユーザー名とパスワードは URL の一部として暗号化されずに渡されます。

HTTP:GET タイプのプロシージャーの場合、URL 句内および (自動的に生成されて) プロシージャーに渡されたパラメーターからの両方でクエリパラメーターを指定できるようになりました。

```
URL 'http://localhost/service?parm=1
```

TYPE 句　Web サービス要求を行う場合に使用するフォーマットを指定します。SOAP が指定されている場合、または TYPE 句が含まれていない場合は、SOAP:RPC が使用されます。HTTP が指定されている場合は、HTTP:POST が使用されます。

TYPE 句では、HTTP:GET、HTTP:POST、HTTP:PUT の各タイプに MIME タイプを指定できます。*MIME タイプ*指定は、Content-Type 要求ヘッダーや操作モードの設定に使用することにより、1 つの呼び出しパラメーターのみで要求の本文を設定できます。パラメーターの置換後に Web サービスのストアドプロシージャー呼び出しを行う場合は、パラメーターがまったくなくなるか、1 つだけ残される場合があります。Web サービスプロシージャーを (置換後に) NULL 値またはパラメーターなしで呼び出すと、本文がなくコンテンツ長が 0 の要求になります。パラメーターの名前と値 (複数のパラメーターが可能) は、HTTP 要求の本文内で URL コード化されます。

一般的な MIME タイプの例を次に示します。

- text/plain
- text/html
- text/xml

TYPE 句のキーワードには次の意味があります。

- **'HTTP:GET'**　デフォルトでは、URL で指定されたパラメーターのエンコード用に、このタイプで MIME タイプ application/x-www-form-urlencoded が使用されます。

  たとえば、クライアントが URL http://localhost/WebServiceName?arg1=param1&arg2=param2 から要求を送信すると、次の要求が生成されます。

  ```
  GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
  // <End of Request - NO BODY>
  ```

○ **'HTTP:POST'**　デフォルトでは、POST 要求の本文で指定されたパラメーターのエンコード用に、このタイプで MIME タイプ application/x-www-form-urlencoded が使用されます。URL パラメーターは、要求の本文に格納されます。

たとえば、クライアントが URL http://localhost/WebServiceName?arg1=param1&arg2=param2 から要求を送信すると、次の要求が生成されます。

```
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 は HTTP:POST と似ていますが、HTTP 要求メソッドが生成されます。HTTP:PUT タイプには、デフォルトのメディアタイプはありません。

次の例は、*%SQLANYSAMP12%¥SQLAnywhere¥HTTP¥put_data.sql* サンプルを実行している SQL Anywhere サーバーにデータをアップロードする、汎用のクライアントプロシージャーを設定する方法を示します。

```
ALTER 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**　サーバーにあるリソースを削除する Web サービスクライアントプロシージャーを設定できます。メディアタイプの指定はオプションです。

次の例は、*put_data.sql* サンプルを実行している SQL Anywhere サーバーからリソースを削除する、汎用のクライアントプロシージャーを設定する方法を示します。

```
ALTER PROCEDURE CDEL(resnm LONG VARCHAR, mediatype LONG VARCHAR)
   URL 'http://localhost/resource/!resnm'
   TYPE 'HTTP:DELETE:!mediatype';

CALL CDEL('hello', 'text/plain' );
```

○ **HTTP:HEAD**　HEAD メソッドは GET メソッドと同じですが、サーバーは本文を返しません。メディアタイプを指定できます。

```
ALTER PROCEDURE CHEAD(resnm LONG VARCHAR)
   URL 'http://localhost/resource/!resnm'
   TYPE 'HTTP:HEAD';

CALL CHEAD( 'hello' );
```

○ **'SOAP:RPC'**　このタイプは、Content-Type ヘッダーを 'text/xml' に設定します。SOAP 操作と SOAP パラメーターは、SOAP エンベロープの XML ドキュメントにカプセル化されます。

○ **'SOAP:DOC'**　このタイプは、Content-Type ヘッダーを 'text/xml' に設定します。SOAP:RPC タイプに似ていますが、より多くのデータ型を送信できます。SOAP 操作と SOAP パラメーターは、SOAP エンベロープの XML ドキュメントにカプセル化されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

TYPE 句に MIME タイプを指定すると、その MIME タイプに Content-Type ヘッダーが自動的に設定されます。

HEADER 句　HTTP Web サービスクライアントプロシージャーを作成する場合は、この句を使用して、HTTP 要求ヘッダーのエントリを追加、変更、または削除します。ヘッダーの仕様は、RFC2616 Hypertext Transfer Protocol − HTTP/1.1 および RFC822 Standard for ARPA Internet Text Messages に非常に近いものになっています。たとえば、HTTP ヘッダーに指定できるのは印字可能な ASCII 文字のみで、大文字と小文字は区別されません。

ヘッダーは、*header-name*:*value-name* ペアとして定義できます。各ヘッダーとその値は、コロン (**:**) を使用して区切ります。このため、ヘッダーにコロンは使用できません。各ペアを **¥n**, **¥x0d ¥n**、**<LF>** (改行)、または **<CR><LF>**(改行に続くキャリッジリターン) で区切ることで、複数のヘッダーを定義できます。

ヘッダー内の連続する複数のスペースは、単一の空白文字に変換されます。

CERTIFICATE 句　安全な (HTTPS) 要求を行うには、HTTPS サーバーで使用される証明書にクライアントがアクセスできる必要があります。必要な情報は、セミコロンで区切られたキー／値のペアの文字列で指定されます。ファイルキーを使用して証明書のファイル名を指定したり、証明書キーを使用してサーバー証明書を文字列で指定したりできます。ファイルキーと証明書キーを一緒に指定することはできません。次のキーを使用できます。

キー	省略形	説明
file		証明書のファイル名
certificate	cert	証明書自体
company	co	証明書で指定された会社
unit		証明書で指定された会社の部署
name		証明書で指定された通称

証明書は、HTTPS サーバーに対する要求、または安全でないサーバーから安全なサーバーにリダイレクトされる可能性がある要求に対してのみ必要です。PEM でフォーマットされた証明書のみがサポートされています。

CLIENTPORT 句　HTTP クライアントプロシージャーが TCP/IP を使用して通信するポート番号を識別します。これは「送信」TCP/IP 接続をフィルターするファイアウォール経由で接続するためのもので、この用途にかぎり推奨されています。単一のポート番号、ポート番号の範囲、または両方の組み合わせを指定できます。たとえば、CLIENTPORT '85,90-97' を指定できます。

PROXY 句　プロキシサーバーの URI を指定します。クライアントがプロキシを介してネットワークにアクセスする場合に使用します。プロシージャーがプロキシサーバーに接続し、そのプロキシサーバーを介して Web サービスに要求を送信することを示します。

SET 句　HTTP および SOAP のプロトコル固有の動作オプションを指定します。次の表に、サポートされている SET オプションを示します。CHUNK と VERSION は、HTTP プロトコルに適用されます。OPERATION は、SOAP プロトコルに適用されます。

○ **'HTTP(CH[UNK]=*option*)'** (HTTP または SOAP) このオプションでは、チャンクを使用するかどうかを指定します。チャンクを使用すると、HTTP メッセージを複数の部分に分割できます。可能な値は、ON (常にチャンクを使用)、OFF (チャンクは使用しない)、AUTO (内容が 8196 バイトを超えた場合にのみチャンクを使用、ただし自動生成されたマークアップを除く) です。たとえば、次の SET 句を使用すると、チャンクは有効になります。

```
SET 'HTTP(CHUNK=ON)'
```

CHUNK オプションを指定しないときの、デフォルトの動作は AUTO です。チャンクされた要求が AUTO モードのときに、ステータスが 505 (HTTP Version Not Supported) または 501 (Not Implemented) または 411 (Length Required) で失敗した場合、クライアントはチャンクされた転送コーディングなしで要求をリトライします。

チャンクされた転送コーディングによる要求を HTTP サーバーがサポートしない場合は、CHUNK オプションを OFF (チャンクを使用しない) に設定してください。

CHUNK モードは HTTP バージョン 1.1 からサポートされる転送エンコードです。CHUNK を ON に設定する場合、バージョン (VER) が 1.1 に設定される必要がありますが、デフォルトのバージョンとして 1.1 を使用する場合は、VER に何も設定しないでください。

○ **' HTTP(VER[SION]=*ver*)'** (HTTP または SOAP) このオプションでは、HTTP メッセージのフォーマットに使用する HTTP プロトコルのバージョンを指定します。たとえば、次の SET 句は HTTP のバージョンを 1.1 に設定します。

```
SET 'HTTP(VERSION=1.1)'
```

可能な値は 1.0 と 1.1 です。VERSION が指定されていない場合は、次のようになります。

●CHUNK が ON に設定された場合、1.1 が HTTP バージョンとして使用される

●CHUNK が OFF に設定された場合、1.0 が HTTP バージョンとして使用される

●CHUNK が AUTO に設定された場合、クライアントが CHUNK モードで送信しているかどうかによって 1.0 か 1.1 が使用される

○ **'REDIR(COUNT=*count*, STATUS=*status-list*)'** (HTTP または SOAP) **302 Found** および **303 See Other** などの HTTP 応答ステータスコードを使用して、特に HTTP POST を実行した後に Web アプリケーションを新しい URI にリダイレクトします。たとえば、クライアント要求は次のようにすることができます。

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```

web サーバー応答は次のようにすることができます。

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

応答では、クライアントは別の HTTP 要求を新しい URI に送信します。REDIR オプションを使用して、許可されるリダイレクションの最大数および自動的にリダイレクトする HTTP 応答ステータスコードを制御できます。

たとえば、SET 'REDIR(count=3, status=301,307)'では、最大制限の 3 リダイレクションおよび 301 と 307 ステータスのリダイレクションが可能です。302 や 303 などの他のリダイレクションステータスコードが受信されると、エラーが発効されます (SQLCODE -983)。

デフォルトのリダイレクション制限 *count* は 5 です。デフォルトでは、HTTP クライアントプロシージャーがすべての HTTP リダイレクションステータスコード (301、302、303、307) に応答して自動的にリダイレクトします。すべてのリダイレクションステータスコードを許可しないようにするには、SET REDIR(COUNT=0)を使用します。このモードでは、リダイレクション応答はエラーになりません (SQLCODE -983)。その代わりに、HTTP ステータスおよび応答ヘッダーによって結果セットが返されます。これにより、呼び出し側は **Location** ヘッダーに含まれる URI に基づいて、条件付きで要求を再発行できます。

303 See Other ステータスを受信する POST HTTP メソッドを指定する Web サービスプロシージャが GET HTTP メソッドを使用してリダイレクト要求を発行します。

Location ヘッダーには、絶対パスまたは相対パスのいずれかを含むことができます。HTTP クライアントプロシージャーがいずれかを処理します。このヘッダーには、クエリパラメーターを含めることもでき、これらのパラメーターはリダイレクトされたロケーションに転送されます。たとえば、ヘッダーに次のようなパラメーターが含まれる場合、後続の GET または POST にこれらのパラメーターが含まれます。

```
Location: alternate_service?a=1&b=2
```

上の例では、クエリパラメーターは a=1&b=2 です。

○ **' SOAP(OP[ERATION]=*soap-operation-name*)** (SOAP のみ) このオプションでは、SOAP 操作の名前を指定します (作成しているプロシージャーの名前と異なる場合)。OPERATION の値は、リモートプロシージャーコールの名前に似ています。たとえば、login という SOAP 操作を呼び出す accounts_login というプロシージャーを作成する場合は、次のように指定します。

```
CREATE PROCEDURE accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
  SET 'SOAP(OPERATION=login)'
```

OPERATION オプションを指定しない場合、SOAP 操作の名前は、作成するプロシージャーの名前と一致する必要があります。

複数の *protocol-option* 設定を 1 つの SET 句で組み合わせると、次の文のようになります。

```
CREATE PROCEDURE accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
 SET 'HTTP ( CHUNK=ON; VERSION=1.1 ), SOAP( OPERATION=login )'
 ...
```

SOAPHEADER 句 (SOAP フォーマットのみ) SOAP Web サービスをプロシージャーとして宣言する場合は、この句を使用して 1 つ以上の SOAP 要求ヘッダーエントリを指定します。SOAP ヘッダーは、静的定数として宣言したり、代入パラメーターメカニズムを使用して動的に設定したりできます (hd1、hd2 などに IN、OUT、または INOUT パラメーターを宣言)。Web サービスプロシージャーでは、1 つ以上の IN モード代入パラメーターと、1 つの INOUT または OUT 代入パラメーターを定義できます。

次の例は、クライアントが、いくつかのヘッダーエントリを代入パラメーターを使用して送信し、応答 SOAP ヘッダーデータを受信するよう指定する方法を示しています。

```
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 句　(SOAP フォーマットのみ) この句は、SOAP:RPC 要求と SOAP:DOC 要求の両方に通常必要なメソッドネームスペースを示します。要求を処理する SOAP サーバーは、このネームスペースを使用して、SOAP 要求メッセージ本文内のエンティティの名前を解釈します。ネームスペースは、Web サービスサーバーから使用できる SOAP サービスの WSDL (Web Services Description Language) から取得できます。デフォルト値は、プロシージャーの URL のオプションのパスコンポーネントの直前までです。

備考

パラメーター値は、要求の一部として渡されます。使用される構文は、要求のタイプによって決まります。HTTP:GET の場合、パラメーターは URL の一部として渡されます。HTTP:POST 要求の場合、値は要求の本文に置かれます。SOAP 要求へのパラメーターは、常に要求本文にバンドルされます。

パーミッション

RESOURCE 権限が必要です。

別のユーザーのプロシージャーを作成するには、DBA 権限が必要です。

関連する動作

オートコミット。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「ALTER PROCEDURE 文」433 ページ
- 「CALL 文」503 ページ
- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [Web サービス]」555 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [T-SQL]」599 ページ
- 「CREATE PROCEDURE 文 [外部呼び出し]」583 ページ
- 「DROP PROCEDURE 文」720 ページ
- 「SOAP の構造化されたデータ型」『SQL Anywhere サーバー プログラミング』
- 「HTTP 要求と SOAP 要求の構造」『SQL Anywhere サーバー プログラミング』
- 「HTTP 要求ヘッダーの管理」『SQL Anywhere サーバー プログラミング』
- 「チュートリアル：SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』
- 「チュートリアル：RAW サービスでの MIME タイプの操作」『SQL Anywhere サーバー プログラミング』
- 「Web サービスへの変数の指定」『SQL Anywhere サーバー プログラミング』
- 「GRANT 文」782 ページ
- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「Web クライアントアプリケーションの開発」『SQL Anywhere サーバー プログラミング』
- 「remote_idle_timeout オプション」『SQL Anywhere サーバー データベース管理』
- 「句の値に使用する代入パラメーター」『SQL Anywhere サーバー プログラミング』
- 「Web クライアントアプリケーションの開発」『SQL Anywhere サーバー プログラミング』
- 「ClientPort (CPORT) プロトコルオプション (クライアント側のみ)」『SQL Anywhere サーバー データベース管理』
- 「HTTP 要求に失敗しました。ステータスコード '%1'」『エラーメッセージ』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　Adaptive Server Enterprise によってサポートされません。

例

次の例は、FtoC という Web サービスクライアントプロシージャーを作成します。

```
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';
```

CREATE PROCEDURE 文 [T-SQL]

Adaptive Server Enterprise 互換の方法で、データベース内にプロシージャーを作成します。

構文

次の Transact-SQL CREATE PROCEDURE 文のサブセットが SQL Anywhere でサポートされています。

CREATE [**OR REPLACE**]**PROCEDURE** [*owner*.]*procedure_name*
[**NO RESULT SET**]
[[**(**] *@parameter-name data-type* [**=** *default*] [**OUTPUT**], ... [**)**]]
[**WITH RECOMPILE**] **AS** *statement-list*

パラメーター

OR REPLACE 句 OR REPLACE を指定すると、新しいプロシージャーが作成されるか、同じ名前の既存のプロシージャーが置き換えられます。この句によって、プロシージャーの定義は変更されますが、既存のパーミッションは保持されます。使用中のプロシージャーを置き換えようとすると、エラーが返されます。

NO RESULT SET 句 このプロシージャーによって結果セットが返されないことを宣言します。この句は、プロシージャーが結果セットを返さないことを外部環境から知る必要がある場合に役立ちます。

WITH RECOMPILE 句 この句は、Transact-SQL との互換性を保つために実装されていますが、無視されます。SQL Anywhere は、常にデータベース起動後に初めて実行されたプロシージャーを再コンパイルし、コンパイルされたプロシージャーをデータベースが停止するまで保管します。

備考

Transact-SQL と SQL Anywhere 文 (Watcom SQL) の間の次の相違点のリストは、両方のダイアレクトで記述する場合に役立ちます。

- **@ をプレフィクスとする変数名** @ 符号 は Transact-SQL 変数名を示します。Watcom-SQL では、変数には有効な識別子であればどれでも使用でき、@ プレフィクスはオプションです。

- **入出力パラメーター** Watcom SQL プロシージャーパラメーターは、デフォルトでは INOUT であり、IN、OUT、または INOUT としても指定できます。Transact-SQL プロシージャーパラメーターは、デフォルトでは INPUT パラメーターです。OUTPUT キーワードを追加すると、入出力として指定できます。Transact-SQL ダイアレクトには、出力専用のパラメーターはありません。

  Watcom SQL ダイアレクトを使用してパラメーター OUT を宣言すると、出力専用になります。ダイアレクトの混在はおすすめできません。これは、プロシージャー宣言をアップロードしたり、プロシージャー宣言を使用してデータベースを再構築したりする場合に問題が生じるからです。プロシージャー宣言をアップロードしたり、プロシージャー宣言を使用してデータベースを再構築したりすると、再構築プロシージャー宣言は Transact-SQL ダイアレクトとなり、OUTPUT キーワードが使用され、パラメーターは入出力になります。

- **パラメーターのデフォルト値** Watcom SQL のプロシージャーパラメーターには、キーワード DEFAULT を使用してデフォルト値を設定します。Transact-SQL では等号 (=) を使用してデフォルト値を設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **結果セットを返す**　Watcom SQL は RESULT 句を使用して、返される結果セットを指定します。Transact-SQL プロシージャーでは、最初のクエリのカラム名またはエイリアス名が呼び出し環境に返されます。

  次の Transact-SQL プロシージャーは、結果セットが Transact-SQL ストアドプロシージャーから返される方法を示します。

```
CREATE PROCEDURE showdept @deptname varchar(30)
AS
  SELECT Employees.Surname, Employees.GivenName
  FROM Departments, Employees
  WHERE Departments.DepartmentName = @deptname
  AND Departments.DepartmentID = Employees.DepartmentID;
```

  次に、これに対応する Watcom SQL のプロシージャーを示します。

```
CREATE PROCEDURE showdept(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;
```

- **プロシージャー本体**　Transact-SQL プロシージャーの本体は、AS キーワードをプレフィクスとして付けた Transact-SQL 文のリストです。Watcom SQL プロシージャーの本文は、BEGIN と END キーワードで囲まれた複合文です。

パーミッション

RESOURCE 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE FUNCTION 文」563 ページ
- 「CREATE PROCEDURE 文」602 ページ

標準と互換性

- **SQL/2008**　Transact-SQL 拡張。
- **Transact-SQL**　SQL Anywhere は Adaptive Server Enterprise の CREATE PROCEDURE 文構文のサブセットをサポートします。

  SQL Anywhere の Transact-SQL ダイアレクトでは、Transact-SQL の SQL プロシージャーのみがサポートされています。外部プロシージャーを作成するには、Watcom SQL の構文を使用してください。Adaptive Server Enterprise では、NO RESULT SET 句はサポートされていません。Transact-SQL WITH RECOMPILE オプション句があっても無視されます。SQL Anywhere は、常にデータベース起動後に初めて実行されたプロシージャーを再コンパイルし、コンパイルされたプロシージャーをデータベースが停止するまで保管します。

SQL Anywhere では、Transact-SQL プロシージャーのグループはサポートされていません。

CREATE PROCEDURE 文

データベースにユーザー定義 SQL プロシージャーを作成します。

構文

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* | *variable-name*

parameter :
parameter-mode parameter-name data-type [**DEFAULT** *expression*]
| **SQLCODE**
| **SQLSTATE**

parameter-mode :
IN
| **OUT**
| **INOUT**

result-column : *column-name data-type*

パラメーター

外部プロシージャーまたはネイティブプロシージャーを呼び出す永続的なストアドプロシージャーの作成には、さまざまなプログラミング言語を使用できます。PROC は PROCEDURE の同義語として使用できます。

OR REPLACE 句　OR REPLACE を指定すると、新しいプロシージャーが作成されるか、同じ名前の既存のプロシージャーが置き換えられます。この句によって、プロシージャーの定義は変更されますが、既存のパーミッションは保持されます。使用中のプロシージャーを置き換えようとすると、エラーが返されます。

TEMPORARY 句　CREATE TEMPORARY PROCEDURE を指定すると、作成した接続でのみ参照できるストアドプロシージャーになり、接続を削除するとプロシージャーも自動的に削除されます。テンポラリストアドプロシージャーを明示的に削除することもできます。テンポラリストアドプロシージャーに対して ALTER、GRANT、または REVOKE は実行できません。また他の関数とは異なり、テンポラリストアドプロシージャーはカタログやトランザクションログに記録されていません。

テンポラリプロシージャーは、作成者 (現在のユーザー) または指定された所有者のパーミッションで実行されます。テンポラリプロシージャーに所有者を指定できるのは次の場合です。

○テンポラリプロシージャーが永続的なストアドプロシージャー内に作成された場合

○テンポラリプロシージャーと永続的なプロシージャーとで所有者が同じ場合

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

テンポラリプロシージャーの所有者を削除するには、テンポラリプロシージャーを先に削除する必要があります。

読み込み専用データベースに接続し、外部プロシージャーにすることができない場合、テンポラリストアドプロシージャーを作成または削除できます。

たとえば、次のテンポラリプロシージャーは CustRank というテーブルがあればそれを削除します。この例では、テーブル名が一意であり、プロシージャーの作成者がテーブルの所有者を指定しなくても参照できると想定しています。

```
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' );
```

パラメーター　パラメーター名は、カラム名など他のデータベース識別子に対するルールに従って付けてください。これらは有効な SQL データ型にする必要があります。

パラメーターには、IN、OUT、INOUT のいずれかのキーワードをプレフィクスとして付けることができます。これらの値のいずれも指定しない場合、パラメーターはデフォルトで INOUT になります。キーワードには次の意味があります。

○ **IN**　このパラメーターは、プロシージャーに値を与える式です。

○ **OUT**　このパラメーターは、プロシージャーから値を受け取ることがある変数です。

○ **INOUT**　このパラメーターはプロシージャーに値を与え、プロシージャーから新しい値を受け取ることがある変数です。

CALL 文を使ってプロシージャーを実行する場合、必ずしもすべてのパラメーターを指定する必要はありません。CREATE PROCEDURE 文の中にデフォルト値がある場合、不明のパラメーターにデフォルト値を割り当てます。CALL に引数が指定されておらず、デフォルトも設定されていない場合には、エラーが発生します。

SQLSTATE と SQLCODE は、プロシージャーが終了するときに、SQLSTATE または SQLCODE 値を出力する、特別な OUT パラメーターです。SQLSTATE と SQLCODE の特別値は、プロシージャーのリターンステータスのテストを目的として、プロシージャー呼び出しの直後にチェックできます。

SQLSTATE と SQLCODE 特別値は、その次の SQL 文によって修正されます。SQLSTATE と SQLCODE をプロシージャー引数として与えると、リターンコードは変数の中に格納されます。

CREATE OR REPLACE PROCEDURE を指定すると、新しいプロシージャーが作成されるか、同じ名前の既存のプロシージャーが置き換えられます。この句によって、プロシージャーの定義は変更されますが、既存のパーミッションは保持されます。OR REPLACE をテンポラリプロシージャーで使用することはできません。置き換え対象のプロシージャーが使用中の場合は、エラーが返されます。CREATE OR REPLACE PROCEDURE 文が実行されると、接続のためのオープンカーソルが閉じられます。

RESULT 句　RESULT 句は結果セットのカラムの数と型を宣言します。RESULT キーワードに続くカッコで囲まれたリストは、結果カラムの名前と型を定義します。CALL 文が記述されていると、この情報を Embedded SQL DESCRIBE または ODBC SQLDescribeCol が返します。

プロシージャーは、その実行方法に応じて、それぞれカラム数が異なる複数の結果セットを生成する場合があります。たとえば次のプロシージャーは、2 カラムを返す場合も、1 カラムを返す場合もあります。

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
   IF formal = 'n' THEN
      SELECT GivenName
      FROM Employees
   ELSE
      SELECT Surname, GivenName
      FROM Employees
   END IF
END;
```

これらの結果セットプロシージャーは RESULT 句を指定しないで記述するか、Transact-SQL で記述します。これらの使用には、次の制約があります。

- **Embedded SQL**　正しい形式の結果セットを取得するには、結果セットのカーソルが開かれてからローが返されるまでの間に、プロシージャーコールを記述 (DESCRIBE) します。DESCRIBE 文の CURSOR *cursor-name* 句は必須です。

- **ODBC、OLE DB、ADO.NET**　変数結果セットプロシージャーは、これらのインターフェイスを使用するアプリケーションで使用できます。結果セットの記述は、ドライバーまたはプロバイダーによって実行されます。

- **Open Client アプリケーション**　変数結果セットプロシージャーは Open Client アプリケーションで使用できます。

- **Web サービス**　Web サービスはストアドプロシージャーの RESULTS 句を使用して、結果セットのカラムの数と型を指定します。Web サービスは、複数の結果セットを返すプロシージャーや EXECUTE IMMEDIATE を使用する変数結果セットには対応していません。

注意
WITH RESULT SET ON 句を含む EXECUTE IMMEDIATE 文をプロシージャーで使用しており、文から返される結果セットがプロシージャーから返される結果セットと同じである場合には、EXECUTE IMMEDIATE 文の結果セットの最初のカラムのみが返されます。

プロシージャーが結果セットを 1 つしか返さない場合、RESULT 句を使用してください。この句を使用すると、カーソルがオープンした後で ODBC と Open Client のアプリケーションが結果セットを記述し直すのを防ぐことができます。

複数の結果セットを処理するために ODBC は、プロシージャーが定義した結果セットではなく、現在実行中のカーソルを記述します。したがって、ODBC はいつもプロシージャー定義の RESULT 句内で定義されているカラム名を記述するわけではありません。この問題を回避するには、結果セットを生成する SELECT 文でカラムエイリアスを使用します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

NO RESULT SET 句　このプロシージャーによって結果セットが返されないことを宣言します。この句は、プロシージャーが結果セットを返さないことを外部環境から知る必要がある場合に役立ちます。

SQL SECURITY 句　SQL SECURITY 句は、INVOKER (プロシージャーを呼び出すユーザー) または DEFINER (プロシージャーを所有するユーザー) としてプロシージャーが実行されるかどうかを定義します。デフォルトは DEFINER です。

SQL SECURITY INVOKER が指定されている場合は、プロシージャーを呼び出すユーザーごとに注釈を行う必要があるためメモリ使用量が増えます。SQL SECURITY INVOKER が指定されている場合は、呼び出し側としても名前の決定が行われます。このため、すべてのオブジェクト名 (テーブル、プロシージャーなど) を該当する所有者で修飾する場合は注意が必要です。たとえば、user1 が次のプロシージャーを作成するとします。

```
CREATE PROCEDURE user1.myProcedure()
   RESULT( columnA INT )
   SQL SECURITY INVOKER
BEGIN
   SELECT columnA FROM table1;
END;
```

user2 がこのプロシージャーを実行しようとし、テーブル user2.table1 が存在**しない**場合、テーブルルックアップエラーが生じます。さらに、user2.table1 が存在**する**場合は、意図する user1.table1 の代わりにこのテーブルが使用されます。このような状況を防ぐには、文においてテーブル参照を修飾します (単なる table1 ではなく、user1.table1 とします)。

ON EXCEPTION RESUME 句　この句は、Transact-SQL のようなエラー処理を Watcom SQL 構文のプロシージャーで使用可能にします。

ON EXCEPTION RESUME を使用すると、プロシージャーは on_tsql_error オプションの設定に応じたアクションを実行します。on_tsql_error を Conditional (デフォルト) に設定すると、次の文がエラーを処理する場合は実行が継続され、そうでない場合は終了します。

エラー処理文には、次のようなものがあります。

- IF
- SELECT @variable =
- CASE
- LOOP
- LEAVE
- CONTINUE
- CALL
- EXECUTE
- SIGNAL
- RESIGNAL
- DECLARE
- SET VARIABLE

ON EXCEPTION RESUME では、明示的なエラー処理コードを使用しないでください。

AT *location-string* 句　*location-string* に指定されたリモートプロシージャーのプロキシストアドプロシージャーを現在のデータベース上に作成します。AT 句は、*location-string* のフィールド

デリミターとしてセミコロン (;) をサポートします。セミコロンがない場合は、ピリオドがフィールドデリミターです。セミコロンを使用すると、データベースと所有者の各フィールドにファイル名と拡張子を使用できます。

リモートプロシージャーを使用する場合、*variable-name* は SQL 変数 *variable-name* の内容に置き換えられます。これは、CHAR、VARCHAR、LONG VARCHAR 型でなければなりません。AT 句の変数の使用の詳細については、「ディレクトリアクセスサーバーの作成 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』の例 2 を参照してください。

リモートプロシージャーが結果セットを返すことができる場合は、たとえすべてのケースで結果セットを返せるわけではなくても、ローカルプロシージャー定義には RESULT 句を含めてください。

備考

CREATE PROCEDURE 文はデータベースにプロシージャーを作成します。DBA 権限がある ユーザーは、所有者を指定することによって他のユーザーのプロシージャーを作成できます。プロシージャーは CALL 文で呼び出します。

ストアドプロシージャーが結果セットを返す場合、出力パラメーターを設定したり戻り値を返したりすることはできません。

複数のプロシージャーからテンポラリテーブルを参照する場合、テンポラリテーブル定義が矛盾していたり、テーブルを参照する文がキャッシュされていたりすると、問題が発生する可能性があります。

パーミッション

テンポラリプロシージャーを作成するのでないかぎり、RESOURCE 権限が必要です。

外部プロシージャーを作成する場合、および別のユーザーのプロシージャーを作成する場合、DBA 権限が必要です。

関連する動作

オートコミット、テンポラリプロシージャーにも適用。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「ALTER PROCEDURE 文」433 ページ
- 「BEGIN 文」497 ページ
- 「CALL 文」503 ページ
- 「CREATE FUNCTION 文」563 ページ
- 「CREATE PROCEDURE 文 [外部呼び出し]」583 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「CREATE PROCEDURE 文 [T-SQL]」599 ページ
- 「CREATE SERVER 文」618 ページ
- 「DROP PROCEDURE 文」720 ページ
- 「EXECUTE IMMEDIATE 文 [SP]」740 ページ
- 「GRANT 文」782 ページ
- 「プロシージャー、トリガー、ユーザー定義関数、バッチで使用される EXECUTE IMMEDIATE」『SQL Anywhere サーバー SQL の使用法』
- 「SQL データ型」85 ページ
- 「on_tsql_error オプション」『SQL Anywhere サーバー データベース管理』
- 「結果セット」『SQL Anywhere サーバー SQL の使用法』
- 「プロシージャー内でのテンポラリテーブルの参照」『SQL Anywhere サーバー SQL の使用法』
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』
- 「リモートプロシージャーの作成 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』
- 「リモートプロシージャーコール (RPC)」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　CREATE PROCEDURE は SQL/2008 標準のコア機能ですが、SQL Anywhere でサポートされている一部のコンポーネントはオプションの SQL 言語機能です。これらの機能のサブセットを次に示します。

  ○SQL SECURITY 句は、オプションの SQL/2008 言語機能 T324 です。

  ○SQL プロシージャーに LONG VARCHAR、LONG NVARCHAR、または LONG BINARY 値を渡す機能は、SQL/2008 言語機能 T041 です。

  ○CREATE TABLE または DROP TRIGGER などの文を使用して SQL プロシージャー内でスキーマオブジェクトを作成または変更する機能は、SQL/2008 言語機能 T651 です。

  ○EXECUTE IMMEDIATE、PREPARE、および DESCRIBE などの文を使用して SQL プロシージャー内で動的 SQL 文を使用する機能は、SQL/2008 言語機能 T652 です。

  CREATE PROCEDURE 文のいくつかの句は、ベンダー拡張です。これらを以下に示します。

  ○TEMPORARY 句。

  ○ON EXCEPTION RESUME 句。

  ○AT 句。

  ○特定のルーチンパラメーター用のオプションの DEFAULT 句。

○RESULT 句と NO RESULT SET 句。SQL/2008 標準では、RETURNS キーワードが使用されます。

○オプションの OR REPLACE 句。

- **Transact-SQL**　CREATE PROCEDURE は、Adaptive Server Enterprise でサポートされています。

参照

- 「CREATE PROCEDURE 文 [T-SQL]」599 ページ

例

次のプロシージャーは、Employees テーブルに問い合わせ、指定の給与 (*sal*) の指定の割合 (*percentage*) 内にある給与を返します。

```
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 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 Employees E JOIN Departments D ON E.DepartmentID = D.DepartmentID
   WHERE E.Salary BETWEEN sal *( 1 - percentage ) AND sal * ( 1 + percentage );
END;
```

次のプロシージャーは、CASE 文を使用してクエリの結果を分類します。

```
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
   DECLARE prod_name CHAR(20);
   SELECT name INTO prod_name FROM 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'
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    WHEN 'Shorts' THEN
       SET type = 'Shorts'
    ELSE
       SET type = 'UNKNOWN'
    END CASE;
END;
```

次の例は、前の例で作成された ProductType プロシージャーを置き換えます。プロシージャーが置き換えられた後、Tee Shirt と Sweatshirt のパラメーターが更新されます。

```
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 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;
```

次のプロシージャーはカーソルのロー上でカーソルとループを使用して、単一の値を返します。

```
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 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;
```

CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]

パブリケーションを作成します。Mobile Link では、パブリケーションが SQL Anywhere リモートデータベース内の同期データを識別します。SQL Remote では、統合データベース内とリモートデータベース内の両方のレプリケートされたデータがパブリケーションによって識別されます。

構文 1 (Mobile Link の一般的な使用方法)

CREATE PUBLICATION [**IF NOT EXISTS**] [*owner.*] *publication-name*
(*article-definition*, ... **)**

article-definition :
TABLE *table-name* [**(** *column-name*, ... **)**]
[**WHERE** *search-condition*]

構文 2 (Mobile Link のスクリプト化されたアップロード)

CREATE PUBLICATION [**IF NOT EXISTS**] [*owner.*]
WITH SCRIPTED UPLOAD
(*article-definition*, ... **)**

article-definition :
TABLE *table-name* [**(** *column-name*, ... **)**]
[**USING (** [**PROCEDURE**] [*owner.*][*procedure-name*]
FOR UPLOAD { **INSERT** | **DELETE** | **UPDATE** }, ... **)**]

構文 3 (Mobile Link のダウンロードのみパブリケーション)

CREATE PUBLICATION [**IF NOT EXISTS**] [*owner.*] *publication-name*
FOR DOWNLOAD ONLY
(*article-definition*, ... **)**

article-definition : **TABLE** *table-name* [**(** *column-name*, ... **)**]

構文 4 (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*]

パラメーター

IF NOT EXISTS 句　IF NOT EXISTS 句を指定し、指定したパブリケーションがすでに存在する場合、変更は行われず、エラーは返されません。

article-definition　パブリケーションは複数のアーティクルで構成されています。各アーティクルで、パブリケーションに含まれる単一のテーブルのローとカラムが識別されます。CONTAINS クエリ内のすべてのカラム参照は、同じテーブルを参照しなければなりません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

アーティクルにカラム名のリストが含まれている場合、それらのカラムがパブリケーションに含まれます。カラム名がリストに含まれない場合、テーブル内のすべてのカラムがパブリケーションに含まれます。Mobile Link の同期の場合、カラム名がリストされないと、テーブルのプライマリキーのすべてのカラムをリストに含める必要があります。

構文 2 では、スクリプト化されたアップロードを実行するパブリケーションに使用されます。また、アーティクルの記述によって、アップロードの定義に使用するスクリプトも登録します。

構文 3 は、ダウンロードのみのパブリケーションに使用されます。また、アーティクルはダウンロードするテーブルとカラムのみを指定します。

WHERE 句　WHERE 句を使用すれば、テーブルローのサブセットをアーティクルに含めるように定義することができます。

Mobile Link アプリケーションでは、WHERE 句はアップロードに含まれているローに影響します (ダウンロードについては download_cursor スクリプトで定義されます)。Mobile Link の SQL Anywhere リモートデータベースでは、WHERE 句は、アーティクルに含まれているカラムのみを参照できます。また、サブクエリ、変数、または非決定的関数を含めることはできません。

SUBSCRIBE BY 句　SQL Remote の場合、アーティクルに含めるテーブルローのサブセットを定義するには、SUBSCRIBE BY 句を使用する方法があります。この句を使用すると単一のパブリケーション定義を使って複数のサブスクライバーが 1 つのテーブルの別々のローを受信できます。

備考

CREATE PUBLICATION 文はデータベースにパブリケーションを作成します。所有者名を指定すれば、別のユーザーのパブリケーションも作成できます。

Mobile Link では、パブリケーションは SQL Anywhere リモートデータベースでは必須で、Ultra Light データベースではオプションです。このようなパブリケーションとそれに対するサブスクリプションによって、Mobile Link サーバーにアップロードされるデータが決定されます。

Mobile Link パブリケーションのオプションは、CREATE SYNCHRONIZATION SUBSCRIPTION 文または ALTER SYNCHRONIZATION SUBSCRIPTION 文の ADD OPTION 句で設定します。

構文 2 はスクリプト化されたアップロードのパブリケーションを作成します。USING 句を使用して、アップロードの定義に使用するストアドプロシージャーを登録します。各テーブルで、3 つまでのストアドプロシージャー (挿入、削除、更新用のプロシージャーを 1 つずつ) を使用できます。

構文 3 は、トランザクションログファイルなしで同期できるダウンロード専用パブリケーションを作成します。ダウンロードのみのパブリケーションが同期されると、ダウンロードされたローは、リモートデータベースのローに加えられた変更を上書きします。

SQL Remote では、パブリッシュは双方向オペレーションなので、データは統合データベースとリモートデータベースのどちらへでも入力できます。SQL Remote インストール環境では、統合データベースとすべてのリモートデータベースのパブリケーション定義は同じにします。統合データベースで SQL Remote 抽出ユーティリティを実行すると、リモートデータベースで自動的に正しい CREATE PUBLICATION 文が実行されます。

パーミッション

文中で参照されるすべてのテーブルに対する DBA 権限と排他アクセスが必要です。

関連する動作

オートコミット。

参照

- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」435 ページ
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」721 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」647 ページ
- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」458 ページ
- SQL Anywhere Mobile Link クライアント：「パブリケーション」『Mobile Link クライアント管理』
- Ultra Light Mobile Link クライアント：「CREATE PUBLICATION 文 [Ultra Light]」『Ultra Light データベース管理とリファレンス』
- SQL Remote：「パブリケーションとアーティクル」『SQL Remote』
- 「スクリプト化されたアップロード」『Mobile Link クライアント管理』
- 「ダウンロード専用のパブリケーション」『Mobile Link クライアント管理』
- 「ISYSSYNC システムテーブル」1005 ページ
- 「スクリプト化されたアップロードのパブリケーション」『Mobile Link クライアント管理』
- 「パブリケーションの作成」『SQL Remote』
- 「テーブル内の一部のカラムだけをパブリッシュする」『SQL Remote』
- 「SUBSCRIBE BY 句を使用して一部のローのみをパブリッシュする」『SQL Remote』
- 「WHERE 句を使用して一部のローだけをパブリッシュする」『SQL Remote』
- 「テーブル内の一部のローのみをパブリッシュする」『SQL Remote』
- 「プライマリキープールのレプリケート」『SQL Remote』
- 「重複分割」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、2 つのテーブルのすべてのカラムとローをパブリッシュします。

```
CREATE PUBLICATION pub_contact (
  TABLE Contacts,
  TABLE Company
);
```

次の文は、1 つのテーブルの一部のカラムのみをパブリッシュします。

```
CREATE PUBLICATION pub_customer (
  TABLE Customers ( ID, CompanyName, City )
);
```

次の文は、Customers テーブルの Status カラムをテストする WHERE 句を組み込んで、アクティブな顧客のローのみをパブリッシュします。

```
CREATE PUBLICATION pub_customer (
  TABLE Customers ( ID, CompanyName, City, State, Status )
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    WHERE Status = 'active'
  );
```

次の文は、subscribe-by 値を与えて一部のローのみをパブリッシュします。この方法を使用できるのは、SQL Remote の場合のみです。

```
CREATE PUBLICATION pub_customer (
  TABLE Customers ( ID, CompanyName, City, State )
  SUBSCRIBE BY State
);
```

subscribe-by 値は、SQL Remote サブスクリプションの作成時に次のように使用されます。

```
CREATE SUBSCRIPTION TO pub_customer ( 'NY' )
  FOR jsmith;
```

次の例は、スクリプト化されたアップロードを使用する Mobile Link パブリケーションを作成します。

```
CREATE PUBLICATION pub WITH SCRIPTED UPLOAD (
    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
    ),
    TABLE t2 AS my_t2 USING (
      PROCEDURE my.t2_ui FOR UPLOAD INSERT
    )
  );
```

次の例は、ダウンロードのみのパブリケーションを作成します。

```
CREATE PUBLICATION p1 FOR DOWNLOAD ONLY (
  TABLE t1
);
```

CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]

データベースからの出力メッセージのメッセージリンクとリターンアドレスを識別します。

構文

CREATE REMOTE MESSAGE TYPE *message-system*
[**ADDRESS** *address-string*]

message-system :
FILE
| **FTP**
| **SMTP**

パラメーター

message-system SQL Remote がサポートするメッセージシステムの 1 つ。次のいずれかの値を指定します。**FILE**、**FTP**、または **SMTP**。

address-string 指定したメッセージシステムに対して有効なアドレスを含む文字列。

備考

Message Agent は、サポートされたメッセージリンクのうちの 1 つを使用して、データベースから出力メッセージを送信します。リモートデータベースが抽出ユーティリティで作成されていれば、指定したリンクを使用するユーザーのリターンメッセージは、指定したアドレスに送られます。Message Agent は、リンク用のリモートユーザーがある場合にのみ、それらのリンクを開始します。

アドレスには、指定したメッセージシステムに応じた、パブリッシャーのアドレスを指定します。電子メールシステムの場合、アドレス文字列には有効な電子メールアドレスを指定します。ファイル共有システムの場合、アドレス文字列には SQLREMOTE 環境変数で設定されているディレクトリのサブフォルダーを指定します。SQLREMOTE 環境変数でディレクトリが指定されていない場合は、現在のディレクトリを指定してください。この設定は、リモートデータベースで GRANT CONSOLIDATE 文を使って無効にできます。

アドレスを削除するには、CREATE REMOTE MESSAGE TYPE 文を ADDRESS 句なしで実行します。

dbinit ユーティリティは、アドレスを除くメッセージタイプを自動的に作成します。CREATE REMOTE MESSAGE TYPE 文は、他の CREATE 文と異なり指定したメッセージタイプがすでに存在してもエラーを返さずに既存のタイプを変更します。

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「GRANT PUBLISH 文 [SQL Remote]」 777 ページ
- 「GRANT REMOTE 文 [SQL Remote]」 780 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」 776 ページ
- 「DROP REMOTE MESSAGE TYPE 文 [SQL Remote]」 722 ページ
- 「ALTER REMOTE MESSAGE TYPE 文 [SQL Remote]」 436 ページ
- 「SQL Remote メッセージシステム」『SQL Remote』
- 「メッセージタイプの作成」『SQL Remote』
- 「FILE メッセージシステム」『SQL Remote』
- 「FTP メッセージシステム」『SQL Remote』
- 「SMTP メッセージシステム」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

抽出ユーティリティを使用してリモートデータベースを抽出した場合は次の文を使用するとファイルメッセージシステムのメッセージ受信者すべてがメッセージを *company* サブフォルダーに送り返すように設定できます。

また、dbremote に対して、*company* サブフォルダーで着信メッセージを確認するように指定します。

```
CREATE REMOTE MESSAGE TYPE file
ADDRESS 'company';
```

CREATE SCHEMA 文

データベースユーザーに対して、テーブル、ビュー、パーミッションのコレクションを作成します。

構文

CREATE SCHEMA
AUTHORIZATION *userid*
[*create-table-statement*
| *create-view-statement*
| *grant-statement*
] ... ;

備考

CREATE SCHEMA 文はスキーマを作成します。スキーマとは、テーブル、ビュー、関連パーミッションのコレクションのことです。

userid には、現在の接続のユーザー ID を指定します。別のユーザーに対するスキーマを作成することはできません。

CREATE SCHEMA 文に含まれるいずれかの文にエラーが発生すると、CREATE SCHEMA 文全体がロールバックされます。

CREATE SCHEMA 文を使用すると、個別の CREATE と GRANT 文を 1 つにまとめて一度に処理することができます。データベース内に SCHEMA データベースオブジェクトは作成されません。オブジェクトを削除するには、個別の DROP TABLE または DROP VIEW 文を使用します。パーミッションを取り消すには、付与されているそれぞれのパーミッションに対して REVOKE 文を使用します。

個々の CREATE または GRANT 文は、文デリミターで区切りません。文デリミターは CREATE SCHEMA 文自身の末尾を区切ります。

それぞれの CREATE または GRANT 文は、まずオブジェクトを作成してから、それにパーミッションを付与するという順番に並べます。

ユーザーに複数のスキーマを作成することはできますが、おすすめしません。

パーミッション

RESOURCE 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE TABLE 文」651 ページ
- 「CREATE VIEW 文」681 ページ
- 「GRANT 文」782 ページ

標準と互換性

- **SQL/2008**　CREATE SCHEMA は、SQL/2008 標準のコア機能です。単一ユーザーに対して複数スキーマを作成する機能は、オプションの SQL/2008 言語機能 F171 です。SQL Anywhere では、CREATE SCHEMA 文内での REVOKE 文の使用をサポートされていません。また、Transact-SQL バッチまたはプロシージャー内での使用も許可されていません。

- **Transact-SQL**　CREATE SCHEMA 文で GRANT 文と REVOKE 文をサポートする Adaptive Server Enterprise でサポートされています。

例

次の CREATE SCHEMA 文は、2 つのテーブルがあるスキーマを作成します。文は、RESOURCE 権限を持つユーザー ID sample_user で実行します。テーブル t2 を作成する文が失敗すると、どちらのテーブルも作成されません。

```
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY )
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

次の CREATE SCHEMA 文の文デリミターは、最初の CREATE TABLE 文の後に配置されます。文デリミターは CREATE SCHEMA 文の終わりをマークするので、例は、データベースサーバーによって 2 つの文バッチとして解釈されます。テーブル t2 を作成する文が失敗しても、テーブル t1 は作成されます。

```
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY );
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

CREATE SEQUENCE 文

複数のテーブル間でユニークな値となるプライマリキー値、およびテーブルのデフォルト値を生成するために使用できるシーケンスを作成します。

構文

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**]

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

OR REPLACE 句　OR REPLACE を指定すると、新しいシーケンスが作成されるか、同じ名前の既存のシーケンスが置き換えられます。OR REPLACE 句を使用しなかった場合、現在のユーザーに対してすでに存在するシーケンスの名前を指定すると、エラーが返されます。

INCREMENT BY 句　最後に割り当てられた値から次のシーケンス値までの増分量を定義します。デフォルトは 1 です。負の値を指定すると、降順のシーケンスが生成されます。INCREMENT BY 値が 0 の場合は、エラーが返されます。

START WITH 句　シーケンスの開始値を定義します。START WITH 句に値を指定しない場合、昇順シーケンスでは MINVALUE が使用され、降順シーケンスでは MAXVALUE が使用されます。START WITH の値が MINVALUE または MAXVALUE で指定された範囲外の場合は、エラーが返されます。

MINVALUE 句　シーケンスで生成される最小値を定義します。デフォルトは 1 です。MINVALUE が (2^63-1) より大きいか -(2^63-1) より小さい場合は、エラーが返されます。また、MINVALUE が MAXVALUE より大きい場合も、エラーが返されます。

MAXVALUE 句　シーケンスで生成される最大値を定義します。デフォルトは -1 です。MAXVALUE が 2^63-1 より大きいか -(2^63-1) より小さい場合は、エラーが返されます。

CACHE 句　より速くアクセスできるようにメモリに保持される、事前に割り付けられたシーケンス値の数を指定します。キャッシュが不足すると、シーケンスキャッシュが再移植され、対応するエントリがトランザクションログに書き込まれます。キャッシュの現在の値は、チェックポイントの時点で ISYSSEQUENCE システムテーブルに転送されます。デフォルトは 100 です。

CYCLE 句　最大値または最小値に達した後に、値の生成を継続するかどうかを指定します。

デフォルトは NO CYCLE です。この場合、最大値または最小値に達すると、エラーが返されます。

備考

シーケンスは、数値の自動生成が可能なデータベースオブジェクトです。シーケンスは特定またはユニークなテーブルカラムにバインドされず、シーケンスが適用されるテーブルカラム経由でのみアクセスできます。

シーケンスでは、次のいずれかの方法で値を生成できます。

- 単調な増加または減少を無制限に行う
- ユーザー定義の制限まで単調な増加または減少を行い、停止する
- ユーザー定義の制限まで単調な増加または減少を行い、循環して始めに返り、再び開始する

CYCLE 句を使用して、シーケンスで値を使い果たした場合の動作を制御します。

シーケンスが増加して MAXVALUE を上回り、CYCLE が指定されている場合には、次のシーケンス値として MINVALUE が使用されます。シーケンスが減少して MINVALUE を下回り、CYCLE が指定されている場合には、次のシーケンス値として MAXVALUE が使用されます。CYCLE が指定されていない場合は、エラーが返されます。

ビュー定義またはマテリアライズドビュー定義では、シーケンス値を使用できません。

パーミッション

RESOURCE 権限が必要です。

関連する動作

なし

参照

- 「ALTER SEQUENCE 文」437 ページ
- 「DROP SEQUENCE 文」722 ページ
- sequence-expression 句、SELECT 文903 ページ
- 「ユニークな値を生成するためのシーケンスの使用」『SQL Anywhere サーバー SQL の使用法』
- 「シーケンス値と AUTOINCREMENT 値からの選択」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　シーケンスは、SQL/2008 言語機能 T176 を構成します。SQL Anywhere では、シーケンスのデータ型にオプションの指定はできません。これを行うには、シーケンスで CAST を使用します。

  また、以下はベンダー拡張です。

  - CACHE 句
  - OR REPLACE 構文
  - CURRVAL 式
  - DEFAULT 式でのシーケンスの使用

例

次の例は、4 で開始して 2 ずつ増加し、循環せず、15 個の値を一度にキャッシュする、Test という名前のシーケンスを作成します。

```
CREATE SEQUENCE Test
START WITH 4
INCREMENT BY 2
NO MAXVALUE
NO CYCLE
CACHE 15;
```

CREATE SERVER 文

リモートサーバーを作成します。

構文 1

CREATE SERVER *server-name*
CLASS *server-class-string*
USING *connection-info-string*
[**READ ONLY**]

server-class-string :
'SAODBC'

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| **'ASEODBC'**
| **'DB2ODBC'**
| **'IQODBC'**
| **'MSSODBC'**
| **'ORAODBC'**
| **'MSACCESSODBC'**
| **'MYSQLODBC'**
| **'ULODBC'**
| **'ADSODBC'**
| **'ODBC'**
| **'SAJDBC'**
| **'ASEJDBC'**
| **'IQJDBC'**

connection-info-string :
{ *host-name*:*port-number* [*/dbname*] | *data-source-name* | *sqlanywhere-connection-string* }

構文 2

CREATE SERVER *server-name*
CLASS 'DIRECTORY'
USING *using-string*

using-string :
'ROOT = *path*
[**;SUBDIRS =** *n*]
[**;READONLY = { YES | NO }]'**
[**;CREATEDIRS = { YES | NO }]'**

パラメーター

- **CLASS 句**　リモート接続に使用するサーバークラスを指定します。サーバークラスには、詳細なサーバー機能情報が入っています。構文 2 では DIRECTORY クラスを使用してローカルコンピューターのディレクトリにアクセスしています。

- **USING 句**　構文 1 では、USING 句によってデータベースサーバーに接続文字列を提示します。適切な接続文字列は使用されるドライバーによって決まり、ドライバーは *server-class-string* によって決まります。

  ODBC ベースのサーバークラスを使用する場合、USING 句は、ODBC データソース名を示す 'dsn=data-source-name' か UNIX のドライバーバイナリまたは Windows のドライバー定義を示す 'driver=driver-name' またはその両方を含むことができる ODBC 接続文字列です。

  SQL Anywhere リモートサーバー (SAODBC サーバークラス) の場合、*connection-info-string* パラメーターには任意の有効な SQL Anywhere 接続文字列を指定できます。任意の SQL Anywhere 接続パラメーターを使用できます。たとえば、接続に問題がある場合は、LOG 接続パラメーターを含めて接続試行をトラブルシューティングできます。

  USING 句には {} で囲んだローカル変数名またはグローバル変数名を含めることもできます。たとえば、'dsn={@mydsn}' を含む USING 句は、*@mydsn* が SQL 変数で、*@mydsn* 変数の現在の内容が USING 句の展開時に置き換えられることを示します。

  SQL 変数 *variable-name* は、CHAR、VARCHAR、または LONG VARCHAR データ型である必要があります。

USING 句の変数の使用の詳細については、「ディレクトリアクセスサーバーの作成 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』の例 2 を参照してください。

JDBC ベースのサーバークラスを使用する場合、USING 句の形式は *host-name*:*port-number* [/ *dbname*] となります。

○ ***host-name*** リモートサーバーが稼働しているコンピューター。

○ ***port-number*** リモートサーバーが受信する TCP/IP ポート番号。SQL Anywhere のデフォルトポート番号は 2638 です。

○ ***dbname*** SQL Anywhere リモートサーバーでは、*dbname* を指定しない場合、デフォルトデータベースが使用されます。Adaptive Server Enterprise の場合は、デフォルトが master データベースです。*dbname* を使用しない場合は、他の方法 (FORWARD TO 文など) を使って別のデータベースを指定します。

構文 2 では、USING クラスでローカルディレクトリに次の値を指定しています。

- **ROOT 句** このパスはデータベースサーバーに対する相対パスであり、ディレクトリアクセスクラスのルートです。ディレクトリアクセスサーバー名を使用してプロキシテーブルを作成すると、プロキシテーブルのパスはこのルートパスに対する相対パスになります。

- **SUBDIRS 句** データベースサーバーがアクセスできるルート内のディレクトリレベル数を表す 0 ～ 10 の数。SUBDIRS を省略したときまたは 0 に設定したときは、ディレクトリアクセスサーバー経由でルートディレクトリのファイルのみにアクセスできます。ディレクトリアクセスサーバー経由で使用できるディレクトリまたはサブフォルダーのいずれかにプロキシテーブルを作成できます。

- **READONLY 句** ディレクトリアクセスサーバーからアクセスするファイルが READONLY であり修正できないかどうかを指定します。デフォルトでは NO に設定されています。

- **CREATEDIRS 句** ディレクトリアクセスサーバーを使用してディレクトリを作成できるかどうかを指定します。デフォルトは NO です。

備考

リモートサーバーを作成すると、ISYSSERVER システムテーブルに追加されます。

- **構文 1** CREATE SERVER 文はリモートサーバーを定義します。

SAJDBC、ASEJDBC、および IQJDBC JDBC ベースのサーバークラスは廃止されているため、使用しないでください。

SQL Anywhere リモートサーバーの定義時に ODBC ドライバーマネージャーをバイパスするには、以下の構文を使用し、*connection-info-string* の残りの部分を後ろに続けます。

```
CREATE SERVER remote-server CLASS 'SAODBC' USING 'DRIVER=SQL Anywhere
Native;DSN=my-dsn';
```

この構文を使用すると、リモートデータアクセスで SQL Anywhere ODBC ドライバーを直接ロードできます。この構文は Windows と UNIX でサポートされます。SQL Anywhere ODBC

ドライバーを直接ロードすると、現在のサーバーのバージョンの ODBC ドライバーが使用されるようになります。また、SQL Anywhere ODBC ドライバーをリモートデータアクセスにのみ使用する場合は、登録する必要はありません。

> **注意**
> アプリケーションで SQL Anywhere 以外のリモートサーバーも使用している場合、または 'DRIVER=SQL Anywhere Native' を使用しないで定義されている SQL Anywhere リモートサーバーがある場合、リモートデータアクセスは他のリモートサーバーのドライバーマネージャーを引き続き使用します。

UNIX プラットフォームでは、提供されている iAnywhere Solutions ODBC ドライバーを使用して ODBC ドライバーマネージャーを参照することもできます。構文は次のとおりです。

```
USING 'driver=SQL Anywhere 12;DSN=my-dsn'
```

- **構文 2**　CREATE SERVER 文を使用すると、データベースサーバーが稼働しているコンピューターのローカルディレクトリ構造にアクセスするディレクトリアクセスサーバーを作成できます。ディレクトリアクセスサーバーを使用する必要があるデータベースユーザーごとに、外部ログインを作成する必要があります。UNIX では、データベースサーバーは特定のユーザー権限で実行されているため、ファイルパーミッションはデータベースサーバーユーザーに付与されているパーミッションに基づきます。

パーミッション

DBA 権限。

Windows Mobile ではサポートされません。

関連する動作

オートコミット。

参照

- 「ALTER SERVER 文」439 ページ
- 「DROP SERVER 文」723 ページ
- 「ISYSSERVER システムテーブル」1004 ページ
- 「CREATE EXTERNLOGIN 文」548 ページ
- 「CREATE EXISTING TABLE 文」546 ページ
- 「CREATE SERVER 文の USING パラメーター」『SQL Anywhere サーバー SQL の使用法』
- 「接続パラメーター」『SQL Anywhere サーバー データベース管理』
- 「リモートデータアクセスのサーバークラス」『SQL Anywhere サーバー SQL の使用法』
- 「ディレクトリアクセスサーバー」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、SQL Anywhere ODBC ドライバーを使用して、testsa という名前の SQL Anywhere リモートサーバーを作成します。

```
CREATE SERVER testsa
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere 12;DSN=remoteSA';
```

次の例は、ODBC ドライバーマネージャーを使用しないで、SQL Anywhere ODBC ドライバーを直接ロードします。

```
CREATE SERVER testsa
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere Native;DSN=remoteSA';
```

次の例は、ASE ODBC ドライバーを使用して、ase_prod という名前の Adaptive Server Enterprise (ASE) リモートサーバーを作成します。

```
CREATE SERVER ase_prod
CLASS 'ASEODBC'
USING 'DSN=remoteASE';
```

次の例は、Oracle サーバー oracle723 のリモートサーバーを作成します。ODBC データソース名は oracle723 です。

```
CREATE SERVER oracle723
CLASS 'ORAODBC'
USING 'oracle723';
```

次の例は、ディレクトリ *c:¥temp* 内のファイルのみを表示するディレクトリアクセスサーバーを作成します。

```
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 permissions, file_name, size FROM diskdir0;
```

次の例は、9 レベルのディレクトリを表示するディレクトリアクセスサーバーを作成します。

```
-- Create a directory server that sees 9 levels of directories.
CREATE SERVER diskserver9
CLASS 'directory'
USING 'ROOT=c:¥temp;SUBDIRS=9';
CREATE EXTERNLOGIN DBA TO diskserver9;
CREATE EXISTING TABLE diskdir9 AT 'diskserver9;;;.';
```

CREATE SERVICE 文 [HTTP Web サービス]

新しい HTTP Web サービスを作成します。

構文

CREATE SERVICE *service-name*
TYPE { **'RAW'** | **'HTML'** | **'JSON'** | **'XML'** }
[**URL** [**PATH**] { **ON** | **OFF** | **ELEMENTS** }]
[*common-attributes*]
[**AS** { *statement* | **NULL** }]

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

common-attributes :
[**AUTHORIZATION** { **ON** | **OFF** }]
[**ENABLE** | **DISABLE**]
[**METHODS** '*method*,...']
[**SECURE** { **ON** | **OFF** }]
[**USER** { *user-name* | **NULL** }]

method :
DEFAULT
| **POST**
| **GET**
| **HEAD**
| **PUT**
| **DELETE**
| **NONE**
| *

パラメーター

service-name Web サービス名に使用できるのは、任意の順序の英数字文字、またはスラッシュ (/)、ハイフン (-)、アンダースコア (_)、ピリオド (.)、感嘆符 (!)、チルダ (~)、アスタリスク (*) 、アポストロフィ (')、左カッコ (()、右カッコ (()) です。ただし、スラッシュ (/) は、サービス名の先頭または最後の文字には使用できず、2 つ以上連続 (// など) して使用することもできません。

サービス **root** に名前を付けることができますが、この名前には特別な機能があります。

TYPE 句 各サービスが特定の応答フォーマットを定義する、サービスタイプを識別します。リストされたサービスタイプのいずれかを指定します。デフォルト値はありません。

○ **'RAW'** 結果セットはフォーマットされずにクライアントに送られます。このサービスを利用するには、すべてのコンテンツのマークアップが明示的に指定されている必要があります。マークアップが付けられた現在のコンテンツを含む複雑な動的コンテンツである JavaScript およびイメージは、オンデマンドで生成できます。sa_set_http_header プロシージャーを使用して Content-Type 応答ヘッダーを設定することにより、メディアタイプを指定できます。すべてのブラウザーでマークアップを text/plaina ではなく HTML として表示するには、HTML マークアップを生成するときに Content-Type ヘッダーを 'text/html' に設定することをおすすめします。

○ **'HTML'** 結果セットはテーブルまたはビューの HTML 表現で返されます。

○ **'JSON'** 結果セットは JavaScript Object Notation (JSON) で返されます。JSON サービスは、自動的には JSON 入力を処理しません。(応答内の) データを JSON 形式で表すことのみ行います。JSON は、**application/x-www-form-urlencoded** がサポートされている POST/PUT メソッドを受け付けます。POST/PUT メソッドで **Content-Type: application/json** を指定すると、アプリケーションは http_variable('body') を使用して JSON (要求) コンテンツを取得します。SQL Anywhere は自動的には JSON 入力を解析しません。解析するかどうかはアプリケーションによって決まります。JSON の詳細については、http://www.json.org/を参照してください。

○ **'XML'** 結果セットは XML として返されます。結果セットがすでに XML の場合、それ以上フォーマットは適用されません。XML でない場合は、XML として自動的にフォーマットさ

れます。代替手段としては、sa_set_http_header プロシージャーを使用して text/xml などの有効な Content-Type を設定した FOR XML RAW 句を使用して、RAW サービスから select を返す方法もあります。

URL 句　URL パスを受け入れるかどうか、受け入れる場合にはどのように処理するかを決定します。URL PATH を指定すると、URL と同じ機能があります。

○ **OFF**　URL 要求でサービス名に続いてパスを指定できないことを示します。OFF はデフォルト設定です。たとえば、パス 要素 /aaa/bbb/ccc のため、次の形式は許可されません。

```
http://host-name/service-name/aaa/bbb/ccc
```

Web サービスの作成時に CREATE SERVICE echo URL PATH OFF が指定されたとします。http://localhost/echo?id=1 に似た URL で次の値が生成されます。

```
HTTP_VARIABLE('id') == 1,
HTTP_HEADER('@HTTPQUERYSTRING') == id=1
```

○ **ON**　URL 要求でサービス名に続いてパスを指定できることを示します。**URL** という名前の専用の HTTP 変数を問い合わせることにより、パスの値が返されます。URL パラメーターを明示的に指定するようにサービスを定義したり、HTTP_VARIABLE 関数を使用して取得したりできます。たとえば、次の形式は許可されます。

```
http://host-name/service-name/aaa/bbb/ccc
```

Web サービスの作成時に CREATE SERVICE echo URL PATH ON が指定されたとします。http://localhost/echo/one/two?id=1 に似た URL で次の値が生成されます。

```
HTTP_VARIABLE('id') == 1,
HTTP_VARIABLE('URL') == one/two,
HTTP_HEADER('@HTTPQUERYSTRING') == id=1
```

○ **ELEMENTS**　URL 要求でサービス名に続いてパスを指定できることを示します。**URL1**、**URL2** などの単一のパラメーターキーワードを指定することにより、セグメントでパスが取得されます。各パラメーターは、HTTP_VARIABLE 関数または NEXT_HTTP_VARIABLE 関数を使用して検索できます。これらの繰り返し関数は、可変の数のパス要素を指定できるアプリケーションで使用できます。たとえば、次の形式は許可されます。

```
http://host-name/service-name/aaa/bbb/ccc
```

Web サービスの作成時に CREATE SERVICE echo URL PATH ELEMENTS が指定されたとします。http://localhost/echo/one/two?id=1 に似た URL で次の値が生成されます。

```
HTTP_VARIABLE('id') == 1,
HTTP_VARIABLE('URL1') == one,
HTTP_VARIABLE('URL2') == two,
HTTP_HEADER('@HTTPQUERYSTRING') == id=1
```

10 個までの要素を取得できます。対応する要素が指定されていない場合は、NULL 値が返されます。上記の例では、対応する要素が指定されていないため、HTTP_VARIABLE('URL3') は NULL を返します。

URL の詳細については、「HTTP Web サーバーの参照」『SQL Anywhere サーバー プログラミング』と「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

AUTHORIZATION 句　サービスに接続するときに、ユーザーが基本的な HTTP 認証を通じてユーザー名とパスワードを指定する必要があるかどうかを決定します。デフォルト値は ON です。AUTHORIZATION が OFF の場合、すべてのサービスに AS 句が必要となり、USER 句でユーザーを指定する必要があります。そのユーザーのアカウントとパーミッションを使用して、すべての要求が実行されます。AUTHORIZATION が ON の場合、すべてのユーザーはユーザー名とパスワードを指定する必要があります。オプションとして、USER 句でユーザー名またはグループ名を指定することにより、サービスを使用できるユーザーを制限することもできます。ユーザー名が NULL の場合、既知のユーザーはすべてサービスにアクセスできます。AUTHORIZATION 句を使用すると、Web サービスでデータベース認証とパーミッションを使用して、データベース内のデータへのアクセスを制御できます。

authorization の値が ON の場合、Web サービスに接続しようとしている HTTP クライアントは、基本認証 (RFC 2617) を使用します。基本認証では、ユーザーとパスワードの情報が base-64 エンコーディングを使用して難読化されます。セキュリティを強化するため、HTTPS プロトコルを使用することをおすすめします。

ENABLE 句と DISABLE 句　サービスを使用できるかどうかを指定します。デフォルトでは、サービスの作成時に有効にされます。サービスを作成または変更するときに、ENABLE 句または DISABLE 句を含めることができます。サービスを無効にすると、事実上、サービスがオフラインになります。無効にしたサービスは、後で ENABLE 句を含む ALTER SERVICE 文を使用して有効にできます。無効にされたサービスに対して HTTP 要求を行うと、通常、404 Not Found HTTP ステータスが返されます。

METHODS 句　サービスでサポートされている HTTP メソッドを指定します。有効な値は、DEFAULT、POST、GET、HEAD、PUT、DELETE、NONE です。アスタリスク (*) は、POST、GET、HEAD メソッドを表す省略形として使用できます。これらのメソッドは、RAW、HTML、XML サービスタイプのデフォルトの要求タイプです。すべての HTTP メソッドがすべてのサービスタイプに有効なわけではありません。次の表は、各サービスタイプに適用できる有効な HTTP メソッドの概要を示します。

メソッド値	適用対象のサービス	説明
DEFAULT	すべて	DEFAULT は、指定されたサービスタイプのデフォルトの HTTP メソッドのセットをリセットする場合に使用します。他のメソッド値が存在するリストに含めることはできません。
POST	RAW、HTML、JSON、XML	デフォルトでは有効になっています。
GET	RAW、HTML、JSON、XML	デフォルトでは有効になっています。
HEAD	RAW、HTML、JSON、XML	デフォルトでは有効になっています。

メソッド値	適用対象のサービス	説明
PUT	RAW、HTML、JSON、XML	デフォルトでは有効になっていません。
DELETE	RAW、HTML、JSON、XML	デフォルトでは有効になっていません。
NONE	すべて	NONE は、サービスへのアクセスを無効にするために使用します。
*****	RAW、HTML、JSON、XML	**'POST,GET,HEAD'** と指定することと同じです。

たとえば、次のいずれかの句を使用して、サービスですべての HTTP メソッドタイプがサポートされていることを指定できます。

```
METHODS '*,PUT,DELETE'
METHODS 'POST,GET,HEAD,PUT,DELETE'
```

サービスタイプに対する要求タイプのリストをデフォルトにリセットするには、次の句を使用できます。

```
METHODS 'DEFAULT'
```

SECURE 句　セキュアリスナーまたは非セキュアリスナーでサービスにアクセスできるかどうかを指定します。ON は、HTTPS 接続のみが受け入れられ、HTTP ポートが受信する接続は HTTPS ポートに自動的にリダイレクトされることを示します。OFF は、Web サーバーの起動時に必要なポートが指定されていると、HTTP 接続と HTTPS 接続の両方が受け入れられることを示します。デフォルト値は OFF です。

USER 句　Web サービス要求を実行するパーミッションを持つデータベースユーザーまたはユーザーのグループを指定します。サービスが AUTHORIZATION OFF で設定されている場合は、USER 句を指定する必要があり、AUTHORIZATION ON で指定する必要があります。(デフォルト) 認証を必要とするサービスに対して HTTP 要求を行うと、401 Authorization Required HTTP 応答ステータスになります。この応答に基づいて、ユーザー ID とパスワードを要求するプロンプトが Web ブラウザーで表示されます。

警告
認証が有効になっている場合 (デフォルト) には、USER 句を指定することを強くおすすめします。指定しないと、すべてのユーザーが認証されます。

USER 句は、サービス要求を処理するのに使用できるデータベースユーザーアカウントを制御します。データベースのアクセスパーミッションは、サービスのユーザーに割り当てられた権限に制限されます。

statement　サービスにアクセスしたときに起動する、ストアドプロシージャー呼び出しなどのコマンドを指定します。

statement なしの DISH 以外のサービスに対する HTTP 要求により、URL 内で実行する SQL 式が指定されます。認証が必要ですが、この機能はサーバーを SQL の挿入に対して脆弱にするため、

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

実稼動システムでは使用しないでください。サービス内で文が定義されると、指定された SQL 文はサービスによって実行できる文のみとなります。

通常の Web サービスアプリケーションでは、*statement* を使用して関数またはプロシージャーを呼び出します。ホスト変数をパラメーターとして渡して、クライアント提供の HTTP 変数にアクセスできます。

次の *statement* は、**AuthenticateUser** という名前のプロシージャーに 2 つのホスト変数を渡すプロシージャー コールを示しています。この呼び出しは Web クライアントが **user_name** および **user_password** 変数を供給することを仮定しています。

```
CALL AuthenticateUser ( :user_name, :user_password );
```

関数またはプロシージャーに渡すホスト変数の詳細については、「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

備考

サービス定義は、ISYSWEBSERVICE テーブルに格納され、SYSWEBSERVICE ビューから検査できます。

パーミッション

DBA 権限。

関連する動作

なし

参照

- 「ALTER SERVICE 文 [HTTP Web サービス]」441 ページ
- 「DROP SERVICE 文」724 ページ
- 「ISYSWEBSERVICE システムテーブル」1008 ページ
- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「root Web サービスの作成とカスタマイズ」『SQL Anywhere サーバー プログラミング』
- 「sa_set_http_header システムプロシージャー」1185 ページ
- 「HTTP Web サーバーでの Web サービスアプリケーションの開発」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　ベンダー拡張。

- **Transact-SQL**　CREATE SERVICE は、XML と RAW タイプに関してのみ Adaptive Server Enterprise でサポートされています。

例

次の例は、JSON サービスの作成方法を示します。

-xs (http または https) オプションを指定してデータベースサーバーを起動してから、次の SQL 文を実行してサービスを設定します。

```
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 Employees
END;

CREATE SERVICE "jsonEmployeeList"
   TYPE 'JSON'
   AUTHORIZATION OFF
   SECURE OFF
   USER DBA
   AS CALL ListEmployees();
```

JSON サービスは、AJAX コールバックで簡単に使用できるデータを提供します。

次の SQL 文を実行して、読み取り可能な形式でサービスを提供する HTML サービスを作成します。

```
CREATE SERVICE "EmployeeList"
   TYPE 'HTML'
   AUTHORIZATION OFF
   SECURE OFF
   USER DBA
   AS CALL ListEmployees();
```

Web ブラウザーを使用して、http://localhost/EmployeeList に似た URL を使用してサービスにアクセスします。

CREATE SERVICE 文 [SOAP Web サービス]

HTTP または DISH サービス経由の新しい SOAP を作成します。

構文 1：HTTP を介した SOAP サービス

CREATE 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**

構文 2：DISH サービス

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**
| *****

パラメーター

service-name Web サービス名に使用できるのは、任意の順序の英数字文字、またはスラッシュ (/)、ハイフン (-)、アンダースコア (_)、ピリオド (.)、感嘆符 (!)、チルダ (~)、アスタリスク (*) 、アポストロフィ (')、左カッコ (()、右カッコ ()) です。ただし、スラッシュ (/) は、サービス名の先頭または最後の文字には使用できず、2 つ以上連続 (// など) して使用することもできません。

他のサービスとは異なり、スラッシュ (/) は、DISH サービス名のどこにも使用できません。

サービス **root** に名前を付けることができますが、この名前には特別な機能があります。

TYPE 句 各サービスが特定の応答フォーマットを定義する、サービスタイプを識別します。リストされたサービスタイプのいずれかを指定します。デフォルト値はありません。

○ **'SOAP'** 結果セットは、SOAP エンベロープと呼ばれる XML ペイロードとして返されます。FORMAT 句を使用すると、データのフォーマットをさらに絞り込むことができます。SOAP サービスへの要求は、単なる汎用 HTTP 要求ではなく有効な SOAP 要求である必要があります。SOAP 標準の詳細については、http://www.w3.org/TR/2000/NOTE-SOAP-20000508/を参照してください。

○ **'DISH'** DISH サービス (SOAP ハンドラーを決定) は、GROUP コンテキスト内で SOAP サービスを参照する SOAP 終了ポイントです。また、SOAP クライアントツールキットで使用するために WSDL (Web Services Description Language) を生成することにより、SOAP サービスに対してインターフェイスも公開します。

GROUP 句 GROUP 句が指定されていない DISH サービスは、データベース内で定義されているすべての SOAP サービスを公開します。慣例により、SOAP サービス名は、GROUP 要素と

NAME 要素で構成できます。最後のスラッシュ文字によって、名前がグループから区切られます。たとえば、**'aaa/bbb/ccc'** と定義されている SOAP サービス名は **'ccc'** であり、グループは **'aaa/bbb'** です。この方法で DISH サービスを区切るのは、無効です。代わりに、SOAP 終了ポイントとなる SOAP サービスのグループを指定するために、GROUP 句が適用されます。

注意
有効な XML を生成するため、WSDL ではスラッシュがアンダースコアに変換されます。GROUP 句が指定されていない DISH サービスでは、スラッシュを含む可能性のあるすべての SOAP サービスが公開されるため、このような DISH サービスを使用する場合には注意してください。アンダースコアを含む SOAP サービス名とグループを一緒に使用する場合は、あいまいにならないように注意してください。

DATATYPE 句　SOAP サービスにだけ適用されます。DATATYPE OFF が指定されていると、SOAP の入力パラメーターと応答データは XMLSchema 文字列型として定義されます。ほとんどの場合、実際のデータ型を使用すると、SOAP クライアントで計算を行う前にデータをキャストする必要がなくなるため、この方法をおすすめします。パラメーターのデータ型は、DISH サービスが生成する WSDL のスキーマセクションで公開されます。出力データ型は、データのカラムごとに XML スキーマ型の属性として表されます。

DATATYPE 句に指定できる値を次に示します。

○ **ON**　入力パラメーターと結果セットの応答のデータ型指定を生成します。

○ **OFF**　すべての入力パラメーターと応答のデータ型は、**XMLSchema** 文字列 (デフォルト) になります。

○ **IN**　入力パラメーターに対してのみ実際のデータ型を生成します。応答のデータ型は **XMLSchema** 文字列になります。

○ **OUT**　応答に対してのみ実際のデータ型を生成します。入力パラメーターは **XMLSchema** 文字列になります。

SOAP サービスの詳細については、「チュートリアル : SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

XMLSchema 型と SQL データ型とのマッピングの詳細については、「SOAP のデータ型」『SQL Anywhere サーバー プログラミング』を参照してください。

FORMAT 句　この句は、SOAP クライアントアプリケーションに応答を送信するときの出力フォーマットを指定します。

SOAP サービスフォーマットが SOAP サービスで指定されていない場合は、関連する DISH サービスフォーマットによって指定されます。デフォルトのフォーマットは DNET です。

SOAP サービスのグループにフォーマットルールを活用するには、SOAP 要求を DISH サービス (SQL Anywhere SOAP 終了ポイント) に指示してください (SOAP 操作)。SOAP サービスの FORMAT 指定は、DISH サービスの指定よりも優先されます。DISH サービスのフォーマット指定は、SOAP サービスで FORMAT 句が定義されていない場合に使用されます。どちらのサービスにも FORMAT が指定されていない場合のデフォルトは **'DNET'** です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

次のフォーマットがサポートされます。

○ **'DNET'**　.NET クライアントアプリケーションで使用するために、互換性のある System.Data.DataSet フォーマットで出力されます (デフォルト)。

○ **'CONCRETE'**　この出力フォーマットは、ローとカラムのオブジェクトの配列を表すインターフェイスを生成できるが、DNET フォーマットを使用できないクライアント SOAP ツールキットをサポートするために使用されます。Java クライアントおよび .NET クライアントは、この出力フォーマットを簡単に使用できます。

DISH サービスの WSDL では、特定の出力フォーマットが公開されます。CONCRETE OFF または最後の手段として、1 つ以上の SOAP サービスの CONCRETE フォーマットが **SimpleDataset** で示されます。WSDL を調べると、任意の数のカラムの配列で構成されるローの配列で、**SimpleDataset** が構成されていることがわかります。特定のカラム名およびデータ型は指定されていないため、この表現方法は理想的ではありません。SOAP サービスで、順に RESULT 句を定義するストアドプロシージャーを呼び出すことをお勧めします。このようにして定義された SOAP サービスを公開する DISH サービスは、WSDL を生成するときに結果セットを完全に記述します。

デフォルトでは、**EXPLICIT ON** が使用され、SOAP サービスの結果セットが記述できる場合は、SOAP サービスごとに特定の Dataset エントリが WSDL に含まれます。各エントリ名の先頭には、SOAP サービス名とアンダースコアが付けられます。たとえば、**test** という名前の SOAP サービスは、各カラム要素の XMLSchema 定義が含まれた **test_Dataset** オブジェクト指定を生成します。

EXPLICIT ON が指定されている場合 (デフォルト)、次の条件が満たされると、WSDL は明示的な **DataSet** 要素を記述します。

●CREATE SERVICE 文がストアドプロシージャーを呼び出している

●カラムとデータ型を記述する RESULT 句がストアドプロシージャーで指定されている

EXPLICIT OFF が指定されている場合、WSDL は **SimpleDataset** 要素を記述します。この説明では、カラム数、カラム名またはデータ型は提供されません。

○ **'XML'**　出力は XMLSchema 文字列フォーマットで生成されます。応答としての XML ドキュメントでは、カラムデータを抽出するために SOAP クライアントでさらに処理が必要です。このフォーマットは、ローおよびカラムの配列を表す中間インターフェイスオブジェクトを生成できない SOAP クライアントに適しています。

○ **NULL**　NULL タイプを指定すると、SOAP サービスまたは DISH サービスでデフォルトの動作が使用されます。ALTER SERVICE 文で NULL タイプを使用すると、既存のサービスのフォーマットタイプが上書きされます。

AUTHORIZATION 句　サービスに接続するときに、ユーザーが基本的な HTTP 認証を通じてユーザー名とパスワードを指定する必要があるかどうかを決定します。デフォルト値は ON です。AUTHORIZATION が OFF の場合、SOAP サービスに AS 句が必要となり、USER 句でユーザーを指定する必要があります。そのユーザーのアカウントとパーミッションを使用して、すべての要求が実行されます。AUTHORIZATION が ON の場合、すべてのユーザーはユーザー名とパスワードを指定する必要があります。オプションとして、USER 句でユーザー名またはグルー

プ名を指定することにより、サービスを使用できるユーザーを制限することもできます。ユーザー名が NULL の場合、既知のユーザーはすべてサービスにアクセスできます。
AUTHORIZATION 句を使用すると、Web サービスでデータベース認証とパーミッションを使用して、データベース内のデータへのアクセスを制御できます。

authorization の値が ON の場合、Web サービスに接続しようとしている HTTP クライアントは、基本認証 (RFC 2617) を使用します。基本認証では、ユーザーとパスワードの情報が base-64 エンコーディングを使用して難読化されます。セキュリティを強化するため、HTTPS プロトコルを使用することをおすすめします。

ENABLE 句と DISABLE 句　サービスを使用できるかどうかを指定します。デフォルトでは、サービスの作成時に有効にされます。サービスを作成または変更するときに、ENABLE 句または DISABLE 句を含めることができます。サービスを無効にすると、事実上、サービスがオフラインになります。無効にしたサービスは、後で ENABLE 句を含む ALTER SERVICE 文を使用して有効にできます。無効にされたサービスに対して HTTP 要求を行うと、通常、404 Not Found HTTP ステータスが返されます。

METHODS 句　サービスでサポートされている HTTP メソッドを指定します。有効な値は、DEFAULT、POST、GET、HEAD、NONE です。POST、GET、HEAD メソッドを表す省略形として、アスタリスク (*) を使用できます。SOAP サービスのデフォルトのメソッドタイプは、POST と HEAD です。DISH サービスのデフォルトのメソッドタイプは、GET、POST、および HEAD です。すべての HTTP メソッドがすべてのサービスタイプに有効なわけではありません。次の表は、各サービスタイプに適用できる有効な HTTP メソッドの概要を示します。

メソッド値	適用対象のサービス	説明
DEFAULT	両方	DEFAULT は、指定されたサービスタイプのデフォルトの HTTP メソッドのセットをリセットする場合に使用します。他のメソッド値が存在するリストに含めることはできません。
POST	両方	SOAP ではデフォルトで有効になります。
GET	DISH のみ	DISH ではデフォルトで有効になります。
HEAD	両方	SOAP と DISH でデフォルトで有効になります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メソッド値	適用対象のサービス	説明
NONE	両方	NONE は、サービスへのアクセスを無効にするために使用します。SOAP サービスに適用すると、SOAP 要求がそのサービスに直接アクセスできなくなります。このため、DISH サービスの SOAP 終了ポイント経由で SOAP 操作にアクセスします。 各 SOAP サービスに METHOD **NONE** を指定することをおすすめします。
*	DISH のみ	**'POST,GET,HEAD'** と指定することと同じです。

たとえば、次のいずれかの句を使用して、サービスですべての HTTP を介した SOAP メソッドタイプがサポートされていることを指定できます。

```
METHODS 'POST,HEAD'
```

サービスタイプに対する要求タイプのリストをデフォルトにリセットするには、次の句を使用できます。

```
METHODS 'DEFAULT'
```

SECURE 句　セキュアリスナーまたは非セキュアリスナーでサービスにアクセスできるかどうかを指定します。ON は、HTTPS 接続のみが受け入れられ、HTTP ポートが受信する接続は HTTPS ポートに自動的にリダイレクトされることを示します。OFF は、Web サーバーの起動時に必要なポートが指定されていると、HTTP 接続と HTTPS 接続の両方が受け入れられることを示します。デフォルト値は OFF です。

USER 句　Web サービス要求を実行するパーミッションを持つデータベースユーザーまたはユーザーのグループを指定します。サービスが AUTHORIZATION OFF で設定されている場合は、USER 句を指定する必要があり、AUTHORIZATION ON で指定する必要があります (デフォルト)。認証を必要とするサービスに対して HTTP 要求を行うと、401 Authorization Required HTTP 応答ステータスになります。この応答に基づいて、ユーザー ID とパスワードを要求するプロンプトが Web ブラウザーで表示されます。

警告
認証が有効になっている場合 (デフォルト) には、USER 句を指定することを強くおすすめします。指定しないと、すべてのユーザーが認証されます。

USER 句は、サービス要求を処理するのに使用できるデータベースユーザーアカウントを制御します。データベースのアクセスパーミッションは、サービスのユーザーに割り当てられた権限に制限されます。

statement　サービスにアクセスしたときに起動する、ストアドプロシージャー呼び出しなどのコマンドを指定します。

DISH サービスは、NULL 文または文のいずれかを指定する必要がある唯一のサービスです。SOAP サービスで分を定義する必要があります。他のサービスが NULL 文を持つ場合もありますが、AUTHORIZATION ON で設定した場合だけです。

statement なしの DISH 以外のサービスに対する HTTP 要求により、URL 内で実行する SQL 式が指定されます。認証が必要ですが、この機能はサーバーを SQL の挿入に対して脆弱にするため、実稼動システムでは使用しないでください。サービス内で文が定義されると、指定された SQL 文はサービスによって実行できる文のみとなります。

通常の Web サービスアプリケーションでは、*statement* を使用して関数またはプロシージャーを呼び出します。ホスト変数をパラメーターとして渡して、クライアント提供の HTTP 変数にアクセスできます。

次の *statement* は、**AuthenticateUser** という名前のプロシージャーに 2 つのホスト変数を渡すプロシージャー コールを示しています。この呼び出しは Web クライアントが **user_name** および **user_password** 変数を供給することを仮定しています。

```
CALL AuthenticateUser ( :user_name, :user_password );
```

関数またはプロシージャーに渡すホスト変数の詳細については、「クライアント提供の HTTP 変数とヘッダーのアクセス」『SQL Anywhere サーバー プログラミング』を参照してください。

備考

サービス定義は、ISYSWEBSERVICE テーブルに格納され、SYSWEBSERVICE ビューから検査できます。

パーミッション

DBA 権限。

関連する動作

なし

参照

- 「ALTER SERVICE 文 [SOAP Web サービス]」446 ページ
- 「DROP SERVICE 文」724 ページ
- 「ISYSWEBSERVICE システムテーブル」1008 ページ
- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「root Web サービスの作成とカスタマイズ」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　CREATE SERVICE は、SOAP タイプに関してのみ Adaptive Server Enterprise でサポートされています。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CREATE SPATIAL REFERENCE SYSTEM 文

空間参照系を作成するか、置き換えます。

構文

{ **CREATE** [**OR REPLACE**] **SPATIAL REFERENCE SYSTEM**
|**CREATE SPATIAL REFERENCE SYSTEM IF NOT EXISTS** }
srs-name
[*srs-attribute* [*srs-attribute* ...]

srs-name : *文字列*

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** }
| **POLYGON FORMAT** *polygon-format*
| **STORAGE FORMAT** *storage-format*

srs-id : *整数*

semi-major-axis-length : *数値*

semi-minor-axis-length : *数値*

inverse-flattening-ratio : *数値*

grid-size : *DOUBLE* : 通常は 0 ～ 1

tolerance-distance : *数値*

axis-order : { **'x/y/z/m'** | **'long/lat/z/m'** | **'lat/long/z/m'** }

polygon-format : { **'CounterClockWise'** | **'Clockwise'** | **'EvenOdd'** }

storage-format : { **'Internal'** | **'Original'** | **'Mixed'** }

パラメーター

OR REPLACE 句 OR REPLACE を指定すると、空間参照系がデータベースにまだ存在しない場合は空間参照系が作成され、存在する場合は置き換えられます。使用中の空間参照系を置き換えようとすると、エラーが返されます。OR REPLACE 句を指定しないでデータベースにすでに存在する空間参照系を置き換えようとしても、エラーが返されます。

CREATE SPATIAL REFERENCE IF NOT EXISTS CREATE SPATIAL REFERENCE IF NOT EXISTS を指定すると、指定された名前の空間参照系がデータベースにすでに存在するかどうかがチェックされます。存在しない場合は、データベースサーバーによって空間参照系が作成されます。存在する場合は、追加のアクションは行われず、エラーも返されません。

IDENTIFIED BY 句 この句は、空間参照系の SRID (*srs-id*) を指定するときに使用します。組織で *organization-srs-id* を指定して空間参照系を定義した場合は、*srs-id* をその値に設定してください。

IDENTIFIED BY 句が指定されていない場合は、デフォルトにより、ORGANIZATION 句または DEFINITION 句で定義された *organization-srs-id* に SRID が設定されます。デフォルトの SRID として使用できる *organization-srs-id* がどちらの句にも指定されていない場合は、エラーが返されます。

空間参照系が周知の座標系に基づいておらず、異なる測地線解釈がされている場合は、1000000000 (10 億) に周知の値を加えた値に *srs-id* 値を設定します。たとえば、測地線の空間参照系 WGS 84 (ID 4326) の平面解釈の SRID は、1000004326 になります。

SQL Anywhere で提供される周知の座標系に基づいていない空間参照系には、SRID 0 を除き 2000000000 (20 億) 以上の SRID が付与されます。2000000000 ～ 2147483647 の範囲の SRID 値は SQL Anywhere で予約されているため、この範囲の SRID は作成しないでください。

OGC などの定義機関または他のベンダーで予約されている SRID を選択する可能性を削減するため、0 ～ 32767 (EPSG で予約済み) または 2147483547 ～ 2147483647 の範囲の SRID は選択しないでください。

また、SRID は 32 ビットの符号付き整数として格納されるため、2^{31}-1 または 2147483647 を超える数値は指定できません。

DEFINITION 句 この句は、座標系のデフォルトを設定または上書きするときに使用します。DEFINITION 句以外の句で属性が設定されている場合は、DEFINITION 句の設定内容に関係なく、他の句で指定された値が優先されます。

definition-string は、SQL/MM と OGC で定義されるような、空間参照系の Well Known Text 構文の文字列です。たとえば、次のクエリは WGS 84 の定義を返します。

```
SELECT ST_SpatialRefSys::ST_FormatWKT( definition )
  FROM ST_SPATIAL_REFERENCE_SYSTEMS
  WHERE srs_id=4326;
```

Interactive SQL で戻り値をダブルクリックすると、読みやすい形で値が表示されます。

DEFINITION 句が指定されている場合は、*definition-string* が解析され、属性のデフォルト値の選択に使用されます。たとえば、*definition-string* に、*organization-name* と *organization-srs-id* を定義する AUTHORITY 要素が含められる場合があります。

definition-string のパラメーター値は、SQL 文の句で明示的に設定した値で上書きされます。たとえば、ORGANIZATION 句が指定されていると、*definition-string* の ORGANIZATION の値が上書きされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ORGANIZATION 句　この句は、新しい空間参照系の基となる空間参照系を作成した組織に関する情報を指定するときに使用します。*organization-name* は空間参照系を作成した組織の名前であり、*organization-srs-id* は空間参照系を識別するために組織で使用する数値識別子です。

TRANSFORM DEFINITION 句　この句は、空間参照系に使用する変換の説明を指定するときに使用します。現時点では、PROJ.4 変換のみがサポートされています。たとえば、WGS 84 の *transform-definition-string* は '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs' です。

サポートされていない変換定義を指定すると、エラーが返されます。

変換定義は、空間参照系間でデータを変換するときに ST_Transform メソッドで使用されます。一部の変換には、*transform-definition-string* が定義されていなくてもかまいません。

LINEAR UNIT OF MEASURE 句　この句は、空間参照系の線形測定単位を指定するときに使用します。指定する値は、ST_UNITS_OF_MEASURE システムビューで定義された線形測定単位と一致している必要があります。

この句が指定されておらず、DEFINITION 句で定義されていない場合、デフォルトは METRE です。

事前に定義された測定単位をデータベースに追加するには、sa_install_feature system プロシージャーを使用します。

カスタム測定単位をデータベースに追加するには、CREATE SPATIAL UNIT OF MEASURE 文を使用します。

注意
METRE と METER はいずれも有効なスペルですが、SQL/MM 標準に準拠している METRE を使用することをおすすめします。

ANGULAR UNIT OF MEASURE 句　この句は、空間参照系の角度測定単位を指定するときに使用します。指定する値は、ST_UNITS_OF_MEASURE システムテーブルで定義された角度測定単位と一致している必要があります。

この句が指定されておらず、DEFINITION 句で定義されていない場合、デフォルトは、地理的空間参照系の場合は DEGREE、非地理的空間参照系の場合は NULL です。

角度測定単位は、地理的空間参照系の場合は NULL 以外、非地理的空間参照系の場合は NULL にしてください。

事前に定義された測定単位をデータベースに追加するには、sa_install_feature system プロシージャーを使用します。

カスタム測定単位をデータベースに追加するには、CREATE SPATIAL UNIT OF MEASURE 文を使用します。

TYPE 句　この TYPE 句は、SRS で点と点を結ぶ線を解釈する方法を制御するために使用します。地理的空間参照系では、TYPE 句で ROUND EARTH (デフォルト) または PLANAR のいずれかを指定できます。ROUND EARTH モデルでは、点と点を結ぶ線が大楕円弧として解釈されま

す。地表面の 2 点を指定し、この 2 点および地球の中心と交差するように平面を選択したとします。この平面は地球と交差し、2 点を結ぶ線はこの交差に沿った最短距離になります。

2 点がまったく反対側にある場合、2 点および地球の中心と交差する単一でユニークな平面は存在しません。このような正反対の点を結ぶ線セグメントは無効であり、ROUND EARTH モデルでエラーになります。

ROUND EARTH モデルでは、地球が回転楕円体と見なされ、地球の湾曲に沿った線が選択されます。場合によっては、2 点間の線が x= 経度、y= 緯度の正距円筒図法における直線として解釈される平面モデルの使用が必要となることがあります。

次の例では、青い線は ROUND EARTH モデルで使用される線の解釈を示し、赤い線は対応する PLANAR モデルを示します。

PLANAR モデルは、他の製品で使用される解釈と一致させるために使用できます。また、ROUND EARTH モデルでサポートされていないメソッド (ST_Area、ST_ConvexHull など) には制限があり、一部のメソッドは部分的にのみサポートされている (ST_Distance はポイントジオメトリでのみサポートされる) ため、PLANAR モデルも便利です。円ストリングに基づいたジオメトリは、ROUND EARTH 空間参照系ではサポートされていません。

非地理的 SRS の場合には、タイプを PLANAR にしてください (これが、TYPE 句が指定されておらず、DEFINITION 句が指定されていないか非地理的定義を使用している場合のデフォルトです)。

COORDINATE 句　この句は、空間参照系の次元に境界を指定するときに使用します。*coordinate-name* は、空間参照系で使用される座標系の名前です。非地理的座標系の場合は、*coordinate-name* に x、y、または m を使用できます。地理的座標系の場合は、*coordinate-name* に LATITUDE、LONGITUDE、z、または m を使用できます。

次元で境界を配置しない場合は、UNBOUNDED を指定します。境界の上限と下限を設定するには、BETWEEN 句を使用します。

X 座標と Y 座標には、関連する境界が必要です。地理的座標系の場合は、これらの設定を COORDINATE 句で上書きしないかぎり、デフォルトにより経度座標は -180 ～ 180 度の間、緯度座標は -90 ～ 90 の間になります。非地理的座標系の場合は、X 座標と Y 座標の両方について CREATE 文で境界を指定してください。

LATITUDE と LONGITUDE は、地理的座標系に使用されます。LATITUDE と LONGITUDE の境界は、指定がなければデフォルトにより地球全体になります。

ELLIPSOID 句　ELLIPSOID 句は、タイプ ROUND EARTH の空間参照系の楕円として、地球を表現するときに使用する値を指定するために使用します。DEFINITION 句が指定されている場合は、この句で楕円定義を指定できます。ELLIPSOID 句が指定されていると、このデフォルトの楕円が上書きされます。

地球の自転が引き起こす扁平化により地球の中心から北極または南極までの距離が赤道までの距離より短くなるため、地球は完全な球体ではありません。このため、地球は、半長径 (中心から赤道までの距離) と半短径 (中心から極までの距離) に異なる値を使用する楕円としてモデル化されます。半長径と逆扁平率を使用して楕円を定義するのが最も一般的ですが、代わりに半短径を使用して指定することもできます (たとえば、完全な球形を使用して地球を近似させる場合には、この方法を使用します)。半長径と半短径は空間参照系の線形単位で定義され、逆扁平率 (1/f) は次の式で求められる比率です。

```
1/f = (semi-major-axis) / (semi-major-axis - semi-minor-axis)
```

SQL Anywhere では、地理的空間参照系で距離を計算するときに楕円定義が使用されます。

地理的空間参照系では (DEFINITION 句または ELLIPSOID 句で) 楕円を定義する必要がありますが、非地理的空間参照系では指定しません。

SNAP TO GRID 句　平らな地球 (平面) の空間参照系では、SNAP TO GRID 句を使用して、SQL Anywhere で計算に使用されるグリッドサイズを定義します。デフォルトでは、X と Y の空間の境界におけるすべての点で 12 有効桁数を格納できるように SQL Anywhere でグリッドサイズが選択されます。たとえば、空間参照系の境界が X は -180 ～ 180 の間、Y は -90 ～ 90 の間の場合、グリッドサイズ 0.000000001 (1E-9) が選択されます。

grid-size には、グリッドにスナップした点を、バインド済み空間のすべての点において同じ精度で表現できるだけの、十分に大きい値を指定してください。*grid-size* が小さすぎると、サーバーからエラーが通知されます。

0 に設定すると、グリッドへのスナップは行われません。

曲面の空間参照系では、SNAP TO GRID を 0 に設定してください。

SNAP TO GRID DEFAULT を指定すると、グリッドサイズがデータベースサーバーで使用されるデフォルトに設定されます。

TOLERANCE 句　平らな地球 (平面) の空間参照系では、TOLERANCE 句を使用して、点を比較するときに使用する精度を指定します。2 点間の距離が *tolerance-distance* より小さい場合、その 2 点は同じと見なされます。*tolerance-distance* を指定することにより、入力データまたは制限された内部精度の不正確さに対する許容度を制御できます。デフォルトでは、*tolerance-distance* は *grid-size* と等しくなるように設定されます。

0 に設定すると、2 点が完全に等しい場合にのみ同じであると見なされます。

曲面の空間参照系では、TOLERANCE を 0 に設定してください。

POLYGON FORMAT 句　SQL Anywhere では、まず、多角形を構成するリングの方向によって多角形が解釈されます。定義された点の順序でリングを移動した場合、多角形の内側がリングの左側になります。PLANAR と ROUND EARTH の空間参照系では、同じルールが適用されます。

SQL Anywhere で使用される解釈は一般的ですが、汎用的な解釈ではありません。正反対の方向を使用する製品もあれば、多角形の解釈にリングの方向を使用しない製品もあります。POLYGON FORMAT 句を使用して、必要に応じて入力データと一致する多角形の解釈を選択できます。次の値がサポートされています。

○ **'CounterClockwise'**　入力は SQL Anywhere の内部解釈に従います。リングの方向に従って、多角形の内側が左側になります。

○ **'Clockwise'**　入力は SQL Anywhere の内部解釈と反対になります。リングの方向に従って、多角形の内側が右側になります。

○ **'EvenOdd'**　EvenOdd はデフォルトフォーマットです。EvenOdd を指定すると、リングの方向は無視され、代わりに多角形の内部はリングのネストによって決まります。外部のリングは最大のリングとなり、内部のリングはこのリングの内側の小さなリングとなります。斜線は、リング内の点から、すべてのリングを交差して外側に向かってトレースされます。交差するリングの数が偶数の場合は、外部リングです。奇数の場合は、内部リングです。

STORAGE FORMAT 句　空間データを外部フォーマット (WKT や WKB など) からデータベースに挿入するとき、空間操作のパフォーマンスとセマンティックを向上するため、データベースサーバーでデータが正規化されます。正規化された表現は、元の表現と異なる場合があります (多角形リングの方向や個々の座標に格納される精度など)。空間の等価性は正規化の後でも保持されますが、精度やリング方向など、元の入力特性の中には再現できないものもあります。場合によっては、元の表現を、正規化された表現とともに、または明示的に格納することがあります。

格納内容を制御するには、STORAGE FORMAT 句に続けて、次のいずれかの値を指定します。

○ **'Internal'**　SQL Anywhere は正規化された表現のみを格納します。元の入力特性を再現する必要がない場合にこの値を指定します。これは、平面 (TYPE PLANAR) の空間参照系のデフォルトです。

> **注意**
> Mobile Link を使用して空間データを同期する場合は、代わりに 「Mixed」 を指定してください。Mobile Link では同期時に等価性がテストされるため、データが元のフォーマットである必要があります。

○ **'Original'**　SQL Anywhere は元の表現のみを格納します。元の入力特性を再現できますが、格納された値のすべての操作に対して正規化の手順を繰り返す必要があり、データの操作が遅くなる可能性があります。

○ **'Mixed'**　SQL Anywhere で内部バージョンが格納されます。元のバージョンと異なる場合は、元のバージョンも格納されます。両方のバージョンを格納すると、元の表現特性を再現

でき、格納された値の操作に対して正規化の手順を繰り返す必要はありません。ただし、ジオメトリごとに 2 つの表現が格納される可能性があるため、記憶領域の要件が大幅に増加します。

Mixed は、曲面の空間参照系 (TYPE ROUND EARTH) のデフォルトフォーマットです。

備考

地理空間参照系では、LINEAR と ANGULAR の両方の測定単位を指定できます。非地理空間参照系では、LINEAR 測定単位のみを指定できます。LINEAR 測定単位は、点と領域間の距離を計算するために使用されます。ANGULAR 測定単位は、角度の緯度と経度を解釈する方法を示し、投影座標系の場合は NULL、地理的座標系の場合は NULL 以外になります。

操作から返される派生ジオメトリはすべて正規化されます。

SQL Anywhere 以外のデータベースで同期されるデータを操作する場合は、データの元の特性を保持できるようにするため、STORAGE FORMAT を 'Original' または 'Mixed' のいずれかに設定してください。

パーミッション

DBA 権限を持つか、SYS_SPATIAL_ADMIN_ROLE グループのメンバーである必要があります。

関連する動作

なし

参照

- 「sa_install_feature システムプロシージャー」1106 ページ
- 「CREATE SPATIAL UNIT OF MEASURE 文」642 ページ
- 「ST_UNITS_OF_MEASURE 統合ビュー」1340 ページ
- 「ST_SPATIAL_REFERENCE_SYSTEMS 統合ビュー」1336 ページ
- 「ALTER SPATIAL REFERENCE SYSTEM 文」453 ページ
- 「空間参照系 (SRS) と空間参照系識別子 (SRID)」『SQL Anywhere サーバー 空間データサポート』
- 「空間データの使用」『SQL Anywhere サーバー 空間データサポート』
- 「sa_install_feature システムプロシージャー」1106 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、空間参照系 mySpatialRS を作成します。

```
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=-85.75
  +x_0=500000
  +y_0=500000
  +ellps=GRS80
  +datum=NAD83
  +units=m
  +no_defs;
```

CREATE SPATIAL UNIT OF MEASURE 文

空間測定単位を作成するか、置き換えます。

構文

CREATE [**OR REPLACE**] **SPATIAL UNIT OF MEASURE** *identifier*
TYPE { **LINEAR** | **ANGULAR** }
[**CONVERT USING** *number*]

パラメーター

OR REPLACE 句　OR REPLACE の指定は、空間測定単位を作成するか、既存の空間測定単位を同じ名前で置き換えます。この句は、現在のパーミッションを保持します。使用中の空間単位を置き換えようとすると、エラーが返されます。

TYPE 句　角度 (ANGULAR) または距離 (LINEAR) のどちらに測定単位を使用するかを定義します。

CONVERT USING　ベース単位に関係する空間単位の換算係数。線形単位の場合、ベース単位は METRE です。角度単位の場合、ベース単位は RADIAN です。

備考

CONVERT USING 句は、定義された測定単位の値をベース測定単位 (ラジアンまたはメートル) に換算する方法を定義するために使用します。指定された換算係数を測定値に乗算して、ベース測定単位の値を取得します。たとえば、測定値 512 ミリメートルに換算係数 0.001 を乗算して、測定値 0.512 メートルを取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

距離 (ST_Distance または ST_Length) または領域を計算する場合、空間参照系では常に線形測定単位が使用されます。たとえば、空間参照系の線形測定単位がマイルの場合、使用する領域単位は平方マイルになります。場合によっては、線形測定単位を使用するよう指定するオプションパラメーターが空間メソッドで受け入れられることがあります。たとえば、空間参照系の線形測定単位がマイルの場合、オプションパラメーター 'metre' を使用すると、2 つのジオメトリ間の距離をメートルで取得できます。

```
SELECT geom1.ST_Distance( geom2, 'metre' );
```

投影座標系の場合は、空間参照系の線形単位で X 座標と Y 座標が指定されます。地理的座標系の場合は、空間参照系に関連する角度測定単位で緯度と経度が指定されます。多くの場合、この角度測定単位は度ですが、有効な任意の角度測定単位を使用できます。

sa_install_feature システムプロシージャーを使用して、事前に定義された測定単位をデータベースに追加できます。

パーミッション

DBA 権限を持つか、SYS_SPATIAL_ADMIN_ROLE のメンバーである必要があります。

関連する動作

なし

参照

- 「sa_install_feature システムプロシージャー」1106 ページ
- 「DROP SPATIAL UNIT OF MEASURE 文」725 ページ
- 「空間データの使用」『SQL Anywhere サーバー 空間データサポート』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、Test という空間測定単位を作成します。

```
CREATE SPATIAL UNIT OF MEASURE Test
TYPE LINEAR
CONVERT USING 15;
```

CREATE STATISTICS 文

オプティマイザーが使用するカラムの統計情報を再作成し、ISYSCOLSTAT システムテーブルに格納します。

構文

CREATE STATISTICS *object-name* [**(** *column-list* **)**]

object-name :
table-name | *materialized-view-name* | *temp-table-name*

備考

CREATE STATISTICS 文は、SQL Anywhere がデータベースクエリの最適化に使用するカラムの統計情報を再作成します。また、ベーステーブル、マテリアライズドビュー、ローカルテンポラリテーブル、グローバルテンポラリテーブルで実行できます。プロキシテーブルに統計情報は作成できません。カラムの統計情報には、指定したカラムに関してデータベース内でのデータ分散を反映するヒストグラムが含まれます。デフォルトで、カラムの統計情報は 5 つ以上のローを持つテーブルでは自動的に作成されます。

データベースクエリの変化が激しい場合や、データが不均等に分散していたり、頻繁に変更されたりする場合は、テーブルまたはカラムに対して CREATE STATISTICS 文を実行すると、パフォーマンスを改善できることもあります。

統計情報が既に存在している場合、テーブルが空でなければテーブルのサイズに関係なく、CREATE STATISTICS 文は既存のカラムの統計情報を更新します。テーブルが空の場合は何も実行されません。空のテーブルについてカラムの統計情報が存在する場合、CREATE STATISTICS 文を使用しても変化しません。空のテーブルについてカラムの統計情報を削除するには、DROP STATISTICS 文を実行します。

CREATE STATISTICS の実行プロセスは、テーブルの完全スキャンを実行します。このため、慎重な検討を行ってから CREATE STATISTICS 文を実行してください。

統計情報を削除する場合は、CREATE STATISTICS 文を使用して再作成することをおすすめします。統計情報がない場合、オプティマイザーは非効率的なデータアクセスプランを生成し、データベースパフォーマンスの低下を招くことがあります。

パーミッション

DBA 権限。

関連する動作

実行プランは変化する可能性があります。

参照

- 「オプティマイザーの推定と統計」『SQL Anywhere サーバー SQL の使用法』
- 「DROP STATISTICS 文」727 ページ
- 「LOAD TABLE 文」814 ページ
- 「ISYSCOLSTAT システムテーブル」995 ページ
- 「ヒストグラムユーティリティ (dbhist)」『SQL Anywhere サーバー データベース管理』
- 「sa_get_histogram システムプロシージャー」1089 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、SalesOrderItems テーブルの ProductID カラムに対するカラムの統計情報を更新します。

```
CREATE STATISTICS SalesOrderItems ( ProductID );
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CREATE SUBSCRIPTION 文 [SQL Remote]

パブリケーションに対してユーザーのサブスクリプションを作成します。

構文

CREATE SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*

publication-name : *identifier*

subscription-value : *string*

subscriber-id : *string*

パラメーター

publication-name このパブリケーション名でユーザーのサブスクリプションが作成されます。パブリケーションの所有者を含めることもできます。

subscription-value パブリケーションのサブスクリプション式と照合される文字列。サブスクライバーは、サブスクリプション式がサブスクリプション値と一致するすべてのローを受信します。

subscriber-id パブリケーションに対するサブスクライバーのユーザー ID。統合データベースでは、サブスクリプションをリモートユーザーに作成する場合、リモートユーザーには REMOTE パーミッションが付与されている必要があります。リモートデータベースでは、サブスクリプションを統合ユーザーに作成する場合、そのユーザーには CONSOLIDATED パーミッションが付与されている必要があります。

備考

SQL Remote では、パブリケーションとサブスクリプションは双方向の関係です。統合データベース上のパブリケーションに対してリモートユーザー用のサブスクリプションを作成するときにはリモートデータベース上のパブリケーションの統合ユーザーにもサブスクリプションを作成してください。抽出ユーティリティ (dbxtract) とデータベース抽出ウィザードでは、デフォルトで、適切な PUBLISH パーミッションと CONSOLIDATE パーミッションが、リモートデータベース内のユーザーに付与されます。

subscription-value が指定されている場合、この値がパブリケーションの各 SUBSCRIBE BY 式と比較されます。サブスクライバーは、指定した文字列が式の値と一致するすべてのローを受信します。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「DROP SUBSCRIPTION 文 [SQL Remote]」728 ページ
- 「GRANT REMOTE 文 [SQL Remote]」780 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」776 ページ
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」958 ページ
- 「START SUBSCRIPTION 文 [SQL Remote]」941 ページ
- 「GRANT PUBLISH 文 [SQL Remote]」777 ページ
- 「ISYSSUBSCRIPTION システムテーブル」1004 ページ
- 「サブスクリプション」『SQL Remote』
- 「プライマリキープールのレプリケート」『SQL Remote』
- 「テーブル内の一部のローのみをパブリッシュする」『SQL Remote』
- 「SUBSCRIBE BY 句を使用して一部のローのみをパブリッシュする」『SQL Remote』
- 「WHERE 句を使用して一部のローだけをパブリッシュする」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、パブリケーション pub_sales にユーザー p_chin のサブスクリプションを作成します。サブスクライバーは、サブスクリプション式が値 Eastern と一致するすべてのローを受信します。

```
CREATE SUBSCRIPTION
TO pub_sales ( 'Eastern' )
FOR p_chin;
```

CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link]

SQL Anywhere の同期プロファイルを作成します。

構文

CREATE [OR REPLACE] SYNCHRONIZATION PROFILE *name string*

パラメーター

OR REPLACE 句　CREATE OR REPLACE SYNCHRONIZATION PROFILE を指定すると、指定された同期プロファイルが存在する場合は、その定義が置き換えられます。

name　作成する同期プロファイルの名前を指定します。各プロファイルの名前はユニークにします。

string　次の説明に従って有効なオプション文字列を指定します。オプション文字列は、*option-name=option-value* 形式のエレメントのセミコロンで区切ったリストとして指定します。たとえば subscription=s1;verbosity=high のように指定します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

同期プロファイルは、同期の制御に使用できる同期オプションの名前付きコレクションです。dbmlsync でサポートされる同期プロファイルオプションのリストについては、「Mobile Link 同期プロファイル」『Mobile Link クライアント管理』を参照してください。

ブール値を取るオプションの場合、値を TRUE に設定すると、コマンドラインで対応するオプションを指定したときと同じになります。

TRUE を指定するには、TRUE、ON、1、YES の値を使用できます。

FALSE を指定するには、FALSE、OFF、0、NO の値を使用できます。

拡張オプションの設定には、次の構文を使用します。

```
CREATE SYNCHRONIZATION PROFILE myprofile
's=mysub;e={ctp=tcpip;adr="host=localhost;port=2439"}'
```

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「ALTER SYNCHRONIZATION PROFILE 文 [Mobile Link]」457 ページ
- 「DROP SYNCHRONIZATION PROFILE 文 [Mobile Link]」729 ページ
- 「SYNCHRONIZE 文 [Mobile Link]」952 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]

SQL Anywhere リモートデータベースで、Mobile Link ユーザーとパブリケーションとの間のサブスクリプションを作成します。

構文

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*

パラメーター

subscription-name このサブスクリプションの識別に使用する一意の名前。代わりに名前付きパラメーターを使用することをおすすめします。

TO 句 この句は、パブリケーション名を指定します。

FOR 句 この句は、1 つ以上の Mobile Link ユーザー名を指定します。ユーザー名を複数指定した場合、ユーザーごとに別個のサブスクリプションが作成されます。サブスクリプション名を指定した場合、1 つの Mobile Link ユーザー名のみを指定できます。

ml_username は、Mobile Link サーバーと同期することが許可されているユーザーです。

FOR 句を省略すると、パブリケーションに対するプロトコルタイプ、プロトコルオプション、拡張オプションが設定されます。FOR 句を省略した場合、サブスクリプション名の指定または SCRIPT VERSION 句の使用はできません。

TYPE 句 同期に使用するネットワークプロトコルを指定します。デフォルトのプロトコルは tcpip です。

ADDRESS 句 Mobile Link サーバーのロケーションなどのネットワークプロトコルオプションを指定します。複数のオプションは、セミコロンで区切ります。

OPTION 句 サブスクリプションについて拡張オプションを設定できます。FOR 句を指定しない場合、拡張オプションはパブリケーションのデフォルト設定として機能します。

SCRIPT VERSION 句 この句は、同期中に使用するスクリプトバージョンを指定します。通常は、実装するスキーマを変更するたびに新しいスクリプトバージョンを指定してください。

FOR 句を省略した場合は、SCRIPT VERSION 句を使用できません。

備考

subscription-name が指定されていない場合は、ユニークな名前が生成されます。パブリケーション名がユニークである場合、生成されるサブスクリプション名は、パブリケーションと同じになります。それ以外の場合は、パブリケーション名の末尾に数値が追加されることによって、ユニークな名前 (pub001、pub002 など) が形成されます。

network-protocol、*protocol-options*、*option* は、複数の個所で設定できます。

この文を使用すると、オプションや他の情報が SQL Anywhere の ISYSSYNC システムテーブルに格納されます。データベースの DBA 権限を持つユーザーであれば、パスワードや暗号化の証

明書などの情報を表示できます。このとき考えられるセキュリティ上の問題を回避するために、dbmlsync コマンドラインに関する情報を指定できます。

パーミッション

パブリケーションで参照されるすべてのテーブルに対する DBA 権限と排他アクセス。

関連する動作

オートコミット。

参照

- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」458 ページ
- 「DROP SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」729 ページ
- SQL Anywhere Mobile Link クライアント： 「同期サブスクリプションの作成」『Mobile Link クライアント管理』
- Ultra Light Mobile Link クライアント： 「Ultra Light クライアントの同期設計」『Ultra Light データベース管理とリファレンス』
- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』
- 「Mobile Link SQL Anywhere クライアントの拡張オプション」『Mobile Link クライアント管理』
- 「CommunicationType (ctp) 拡張オプション」『Mobile Link クライアント管理』
- 「ISYSSYNC システムテーブル」1005 ページ
- 「優先順位」『Mobile Link クライアント管理』
- 「スクリプトバージョン」『Mobile Link サーバー管理』
- 「dbmlsync 構文」『Mobile Link クライアント管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、Mobile Link ユーザー ml_user1 とパブリケーション sales_publication との間の sales という名前のサブスクリプションを作成します。サブスクリプションが同期されると、スクリプトバージョン sales_v1 が使用され、テーブルが排他モードでロックされます。

```
CREATE SYNCHRONIZATION SUBSCRIPTION sales
TO sales_publication
FOR ml_user1
OPTION locktables='exclusive'
SCRIPT VERSION 'sales_v1'
```

次の例は、FOR 句を省略し、パブリケーション sales_publication の設定を格納します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO sales_publication
ADDRESS 'host=test.internal;port=2439;
security=ecc_tls'
OPTION locktables='exclusive';
```

CREATE SYNCHRONIZATION USER 文 [Mobile Link]

SQL Anywhere リモートデータベースで Mobile Link ユーザーを作成します。

構文

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*

パラメーター

ml_username Mobile Link ユーザーを識別する名前。

TYPE 句 同期に使用するネットワークプロトコルを指定します。デフォルトのプロトコルは tcpip です。

ADDRESS 句 *protocol-options* を *keyword*=*value* の形式でセミコロンで区切って指定します。どのような設定を指定するかは、使用する通信プロトコル (TCP/IP、TLS、HTTP、HTTPS) に応じて異なります。

OPTION 句 OPTION 句では、*option*=*value* をカンマで区切ったリストで拡張オプションを設定できます。

各オプションの値に、等号とセミコロンは使用できません。データベースサーバーは、入力されたオプションを、妥当性を検査しないですべて受け入れます。そのため、オプションのスペルを間違えたり、無効な値を入力したりしても、dbmlsync コマンドを実行して同期を行うまでエラーメッセージが表示されません。

同期ユーザー用に設定したオプションは、個々のサブスクリプション内で、または dbmlsync コマンドラインを使用して上書きできます。

network-protocol、*protocol-options*、*options* は、複数の個所で設定できます。

この文を使用すると、オプションや他の情報が SQL Anywhere の ISYSSYNC システムテーブルに格納されます。データベースの DBA 権限を持つユーザーであれば、パスワードや暗号化の証明書などの情報を表示できます。このとき考えられるセキュリティ上の問題を回避するために、dbmlsync コマンドラインに関する情報を指定できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「ALTER SYNCHRONIZATION USER 文 [Mobile Link]」461 ページ
- 「DROP SYNCHRONIZATION USER 文 [Mobile Link]」731 ページ
- 「CommunicationType (ctp) 拡張オプション」『Mobile Link クライアント管理』
- 「Mobile Link サーバー／クライアント通信の暗号化」『SQL Anywhere サーバー データベース管理』
- 「ISYSSYNC システムテーブル」1005 ページ
- 「dbmlsync 構文」『Mobile Link クライアント管理』
- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』
- 「Mobile Link SQL Anywhere クライアントの拡張オプション」『Mobile Link クライアント管理』
- 「優先順位」『Mobile Link クライアント管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、TCP/IP 経由でサーバーコンピューター mlserver.mycompany.com と同期を行う、パスワード Sam を持つ Mobile Link ユーザー SSinger を作成します。ユーザー定義でのパスワードの使用には、セキュリティは適用**されません**。

```
CREATE SYNCHRONIZATION USER SSinger
TYPE http
ADDRESS 'host=mlserver.mycompany.com'
OPTION MobiLinkPwd='Sam';
```

CREATE TABLE 文

データベースに新しいテーブルを作成し、オプションでリモートサーバー上にテーブルを作成します。

構文

CREATE [**GLOBAL TEMPORARY**] **TABLE** [**IF NOT EXISTS**] [*owner*.]*table-name*
({ *column-definition* | *table-constraint* | *pctfree* }, ... **)**
[{ **IN** | **ON** } *dbspace-name*]
[**ENCRYPTED**]
[**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS**
  | **NOT TRANSACTIONAL**]

[**AT** *location-string*]
[**SHARE BY ALL**]

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**]
[*column-constraint* ...]

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**
| **LAST USER**
| **NULL**

column-constraint :
[**CONSTRAINT** *constraint-name*] {
UNIQUE [**CLUSTERED**]
| **PRIMARY KEY** [**CLUSTERED**] [**ASC** | **DESC**]
| **REFERENCES** *table-name* [**(** *column-name* **)**]
[**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]
[*action-list*] [**CLUSTERED**]
| **CHECK (** *condition* **)**
}
| **COMPUTE (** *expression* **)**

table-constraint :
[**CONSTRAINT** *constraint-name*] {
UNIQUE [**CLUSTERED**] **(** *column-name* [**ASC** | **DESC**], ... **)**
| **PRIMARY KEY** [**CLUSTERED**] **(** *column-name* [**ASC** | **DESC**], ... **)**
| **CHECK (** *condition* **)**
| *foreign-key-constraint*
}

foreign-key-constraint :
[**NOT NULL**] **FOREIGN KEY** [*role-name*]

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

[(*column-name* [**ASC** | **DESC**], ...)]
REFERENCES *table-name*
[(*column-name*, ...)]
[**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]
[*action-list*] [**CHECK ON COMMIT**] [**CLUSTERED**] [**FOR OLAP WORKLOAD**]

action-list :
[**ON UPDATE** *action*]
[**ON DELETE** *action*]

action :
CASCADE
| **SET NULL**
| **SET DEFAULT**
| **RESTRICT**

location-string :
remote-server-name.[*db-name*].[*owner*].*object-name*
| *remote-server-name*;[*db-name*];[*owner*];*object-name*

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

パラメーター

IN 句　この句は、ベーステーブルが格納される DB 領域を指定するときに使用します。この句が指定されていない場合、ベーステーブルは default_dbspace オプションで指定された DB 領域に作成されます。

テンポラリテーブルはテンポラリ DB 領域だけに作成できます。GLOBAL TEMPORARY テーブルを作成するときに IN を指定すると、テーブルはテンポラリ DB 領域に作成されます。ユーザー定義の DB 領域を指定すると、エラーが返されます。

ENCRYPTED 句　ENCRYPTED 句は、テーブルの暗号化が必要なときに使用します。テーブルを暗号化する場合、データベースの作成時にテーブルの暗号化を有効にする必要があります。その後、データベースの作成時に指定された暗号化キーとアルゴリズムを使用してテーブルが暗号化されます。

ON COMMIT 句　ON COMMIT 句はテンポラリテーブル用にのみ使用します。デフォルトで、テンポラリテーブルのローは COMMIT のときに削除されます。SHARE BY ALL 句を指定する場合、ON COMMIT PRESERVE ROWS または NOT TRANSACTIONAL を指定する必要があります。

NOT TRANSACTIONAL 句　グローバルテンポラリテーブルを作成するときに NOT TRANSACTIONAL 句を使用できます。NOT TRANSACTIONAL を使用して作成されたテーブルは、COMMIT または ROLLBACK の影響を受けません。SHARE BY ALL 句を指定する場合、ON COMMIT PRESERVE ROWS または NOT TRANSACTIONAL を指定する必要があります。

AT 句　*location-string* に指定された異なるサーバーにリモートテーブルを作成し、そのリモートテーブルにマッピングするプロキシテーブルを現在のデータベース上に作成します。AT 句は、*location-string* のフィールドデリミターとしてセミコロン (;) をサポートします。セミコロンが

ない場合は、ピリオドがフィールドデリミターです。この構文を使用すると、データベースと所有者の各フィールドにファイル名と拡張子を使用できます。

たとえば、次の文は、テーブル a1 を Microsoft Access ファイル *mydbfile.mdb* にマッピングします。

```
CREATE TABLE a1
AT 'access;d:¥mydbfile.mdb;;a1';
```

Windows Mobile は、AT 句をサポートしません。

外部キー定義は、リモートテーブルでは無視されます。リモートテーブルを参照するローカルテーブルでの外部キー定義も無視されます。プライマリキー定義は、リモートサーバーにサポートされている場合、そのデータベースサーバーに送信されます。

SHARE BY ALL 句　この句を使用できるのは、グローバルテンポラリテーブルを作成して、データベースに対するすべての接続でテーブルを共有する場合のみです。SHARE BY ALL 句を指定する場合、ON COMMIT PRESERVE ROWS または NOT TRANSACTIONAL を指定する必要があります。

IF NOT EXISTS 句　指定された名前のテーブルがすでに存在する場合、変更は行われず、エラーは返されません。

column-definition　テーブル内のカラムを定義します。次は、カラム定義の一部を示します。

- ***column-name***　カラム名は識別子です。同じテーブル内の 2 つのカラムが同じ名前を持つことはできません。
- ***data-type***　カラムに格納されているデータ型。
- **COMPRESSED 句**　カラムを圧縮します。たとえば、次の文は filename と contents という 2 つのカラムがあるテーブル t を作成します。contents カラムは LONG BINARY であり、圧縮されています。

  ```
  CREATE TABLE t (
    filename VARCHAR(255),
    contents LONG BINARY COMPRESSED
  );
  ```

INLINE 句と PREFIX 句　INLINE 句は、ローに格納する最大 BLOB サイズをバイト単位で指定します。INLINE 句で指定された値以下のサイズの BLOB がローに格納されます。INLINE 句で指定された値を超えるサイズの BLOB は、ロー外のテーブル拡張ページに格納されます。また、BLOB のサイズが INLINE 句で指定された値より大きい場合、BLOB の先頭から数バイト分を複製してローに保持することができます。PREFIX 句は、ローに保持されるバイト数を指定します。ローの受け入れと拒否の判断に BLOB のプレフィクスバイトを必要とする要求のパフォーマンスは、PREFIX 句によって向上する可能性があります。

圧縮されたカラムのプレフィクスデータは圧縮されずに格納されるため、要求を満たすために必要なすべてのデータがプレフィクス内に格納されている場合は、解凍は必要ありません。

INLINE と PREFIX のどちらも指定しない場合、または USE DEFAULT を指定している場合、デフォルト値は次のように適用されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○CHAR、NCHAR、LONG VARCHAR などの文字データ型のカラムの場合、INLINE のデフォルト値は 256 で、PREFIX のデフォルト値は 8 です。

○BINARY、LONG BINARY、VARBINARY、BIT、VARBIT、LONG VARBIT、BIT VARYING、UUID など、バイナリデータ型のカラムの場合、INLINE のデフォルト値は 256 であり、PREFIX のデフォルト値は 0 です。

注意
デフォルト以外の設定が必要な特殊な環境でない限り、デフォルト値を使用するようにしてください。デフォルト値は、パフォーマンスとディスク領域の要件のバランスをとって選択されています。たとえば、INLINE に大きな値を設定し、すべての BLOB をインラインで格納するようにした場合、ローの処理パフォーマンスは低下することがあります。また、PREFIX の値を大きくしすぎると、BLOB の一部を複製したプレフィクスデータのために、BLOB の格納に必要なディスク領域のサイズが増えることになります。

値の 1 つのみを指定する場合、その他の値は指定した値と競合しない最大の値に自動的に設定されます。INLINE 値と PREFIX 値のどちらも、データベースページサイズを超えるサイズを指定できません。また、ローデータの格納に使用できないテーブルページには、小さいサイズですが予約済みのオーバーヘッドがあります。そのため、データベースのページサイズに近い INLINE 値を指定すると、やや少ないバイト数がインラインで格納される可能性があります。

INDEX 句と NO INDEX 句　BLOB の格納時に (文字型またはバイナリ型のみ)、BLOB サイズの内部しきい値 (約 8 データベースページ) を超える挿入値に対して BLOB インデックスを作成する場合は、INDEX を指定します。これはデフォルトの動作です。

BLOB インデックスは、BLOB 内のランダムアクセス検索が必要なときにパフォーマンスを改善する可能性があります。ただし、ランダムアクセスの必要がない画像ファイルやマルチメディアファイルなど、BLOB 値の種類によっては、BLOB インデックスをオフにするとパフォーマンスが改善されることがあります。カラムの BLOB インデックスをオフにするには、NO INDEX を指定します。

注意
BLOB インデックスは、テーブルインデックスとは異なります。テーブルインデックスは、1 つ以上のカラムの値のインデックスとして作成されます。

NULL 句と NOT NULL 句　NULL が指定されると、カラムで NULL 値が許可されるようになります。これはデフォルトの動作です。この設定は、allow_nulls_by_default データベースオプションを使用して制御します。

デフォルトで、BIT として宣言されているカラムは NULL 入力不可ですが、明示的に NULL 入力可にすることができます。

NOT NULL が指定されると、NULL 値は許可されません。

カラムが UNIQUE 制約または PRIMARY KEY 制約の一部である場合、NULL 句が指定されていても、そのようなカラムに NULL を含めることはできません。

DEFAULT 句　DEFAULT 値を指定する場合、カラムの値を指定しない INSERT 文のカラムの値としてこのデフォルト値が使用されます。INSERT 文はカラムの値を指定しません。DEFAULT 値を指定しない場合、これは DEFAULT NULL と同じです。

DEFAULT に指定できる値を次に示します。

○ ***special-value***　DEFAULT 句には複数の特別値のうちの 1 つを使用します。

○ **[CURRENT] TIMESTAMP**　テーブル内の各ローが最後に変更された日付を示すことができます。カラムの宣言に DEFAULT TIMESTAMP が指定されている場合は、ローを挿入するとタイムスタンプのデフォルト値が割り付けられます。この値は、ローが更新されるたびに現在の日付と時刻に基づいて更新されます。

挿入されたローにタイムスタンプのデフォルト値を割り付け、そのローが更新されてもタイムスタンプを更新しない場合は、DEFAULT TIMESTAMP の代わりに DEFAULT CURRENT TIMESTAMP を使用します。

DEFAULT TIMESTAMP で宣言されたカラムにはユニークな値が入ります。これにより、アプリケーションは、ほぼ同時に行われた同じローの更新を検出できます。現在の TIMESTAMP 値が直前の値と同じ場合は、default_timestamp_increment オプションの値が加えられます。

default_timestamp_increment オプションに基づいて、SQL Anywhere の TIMESTAMP 値を自動的にトランケートできます。これは、記録されるタイムスタンプ値の精度が低い他のデータベースソフトウェアとの互換性を維持する場合に便利です。

グローバル変数 @@dbts は、DEFAULT TIMESTAMP を使用するカラムの最後に生成された値を表す TIMESTAMP 値を返します。

○ **[CURRENT] UTC TIMESTAMP**　テーブル内の各ローが最後に変更された日付を示すことができます。カラムの宣言に DEFAULT UTC TIMESTAMP が指定されている場合は、ローを挿入するとタイムスタンプのデフォルト値が割り付けられます。この値は、ローが更新されるたびに現在の協定世界時 (UTC: Coordinated Universal Time) に基づいて更新されます。

挿入されたローにタイムスタンプのデフォルト値を割り付け、そのローが更新されてもタイムスタンプを更新しない場合は、DEFAULT UTC TIMESTAMP の代わりに DEFAULT CURRENT UTC TIMESTAMP を使用します。

このデフォルトの動作は TIMESTAMP や CURRENT TIMESTAMP と同じですが、日付と時刻が協定世界時 (UTC: Coordinated Universal Time) になる点が異なります。

○ **string**　「文字列」5 ページを参照してください。

○ **global-variable**　「グローバル変数」79 ページを参照してください。

○ ***constant-expression***　DEFAULT 句では、データベースオブジェクトを参照していない定数式を使用できます。その結果、GETDATE や DATEADD などの関数を使用できます。式が関数または単純な値でない場合、カッコで囲みます。

○ ***sequence-expression***　データベース内のシーケンスの現在の値または次の値に、DEFAULT を設定できます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **AUTOINCREMENT**　AUTOINCREMENT を使用する場合、カラムは整数データ型の 1 つ、または真数値型にします。

テーブルに挿入する場合、AUTOINCREMENT カラムの値を指定しないと、カラム内の任意の値より大きいユニーク値が生成されます。INSERT によって、カラムの現在の最大値よりも大きなカラム値を指定すると、その値は挿入され、以降の挿入時に開始ポイントとして使用されます。

ローを削除しても AUTOINCREMENT カウンターは減りません。ローの削除によって作成されたギャップは、挿入を行うときに明示的に割り当てることによってのみ埋めることができます。最大値未満のカラム値を明示的に挿入した後、明示的に割り当てられていない次のローを、以前の最大値より 1 大きい値を使ってオートインクリメントさせます。

カラムに直前に挿入された値は、グローバル変数 @@identity を調べることによって確認できます。

AUTOINCREMENT 値は、SYSTABCOL システムビューの max_identity カラムのデータ型に応じて、符号付き 64 ビット整数として保持されます。生成された次の値が、AUTOINCREMENT が割り当てられたカラムに格納できる最大値を超えた場合は、NULL が返されます。NULL を入力できないように宣言されたカラムであると (プライマリキーのカラムである場合など)、SQL エラーが生成されます。

カラムに使用する次の値は、sa_reset_identity プロシージャーを使用してリセットできます。

○ **GLOBAL AUTOINCREMENT**　このデフォルトは、Mobile Link 同期環境または SQL Remote レプリケーションにおいて複数のデータベースを使用するときのために用意されたものです。

このオプションは AUTOINCREMENT と同じですが、ドメインはパーティションに分割されます。各分割には同じ数の値が含まれます。データベースの各コピーにユニークなグローバルデータベース ID 番号を割り当てます。SQL Anywhere では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割からのみ設定されます。

AUTOINCREMENT キーワードの直後にカッコで分割サイズを指定できます。この分割サイズには任意の正の整数を設定できますが、通常、分割サイズは、サイズの値がすべての分割で不足しないように選択されます。

カラムの型が BIGINT または UNSIGNED BIGINT である場合、デフォルトの分割サイズは $2^{32} = 4294967296$ です。それ以外の型のカラムの場合、デフォルトの分割サイズは $2^{16} = 65536$ です。特に、カラムの型が INT または BIGINT ではない場合は、これらのデフォルト値が適切ではないことがあるため、分割サイズを明示的に指定するのが最も賢明です。

このデフォルトを使用する場合、各データベース内のパブリックオプション global_database_id は、ユニークな正の整数に設定します。この値は、データベースをユニークに識別し、デフォルト値の割り当て元の分割を示します。使用できる値の範囲は $np + 1$ ～ $p(n+1)$ です。ここで、n はパブリックオプション global_database_id の値を表し、p は分割サイズを表します。たとえば、分割サイズを 1000、global_database_id を 3 に設定すると、範囲は 3001 ～4000 になります。

前の値が $p(n + 1)$ 未満であれば、このカラム内でこれまで使用した最大値より 1 大きい値が次のデフォルト値になります。カラムに値が含まれていない場合、最初のデフォルト値は np

+ 1 になります。デフォルトのカラム値は、現在の分割以外のカラムの値の影響を受けません。つまり、*np* + 1 より小さいか *p*(*n* + 1) より大きい数には影響されません。Mobile Link または SQL Remote 経由で別のデータベースからレプリケートした場合、このような値になることがあります。

カラムに直前に挿入された値は、グローバル変数 @@identity を調べることによって確認できます。

GLOBAL AUTOINCREMENT 値は、SYSTABCOL システムビューの max_identity カラムのデータ型に応じて、符号付き 64 ビット整数として保持されます。NULL 値は、分割で値が不足したときにも生成されます。NULL を入力できないように宣言されたカラムであると (プライマリキーのカラムである場合など)、SQL エラーが生成されます。この場合には、別の分割からデフォルト値を選択できるように、データベースに global_database_id の新しい値を割り当ててください。未使用の値が残り少ないことを検出し、このような状態を処理するには、GlobalAutoincrement タイプのイベントを作成します。

public オプション global_database_id は、負の値に設定できないため、選択された値は常に正になります。ID 番号の最大値を制限するのは、カラムデータ型と分割サイズだけです。

public オプション global_database_id がデフォルト値の 2147483647 に設定されると、NULL 値がカラムに挿入されます。NULL 値が許可されていない場合に、ローの挿入を試みるとエラーが発生します。

カラムに使用する次の値は、sa_reset_identity プロシージャーを使用してリセットできます。

○ **LAST USER**　LAST USER は、ローを最後に変更したユーザーのユーザー ID です。

LAST USER は文字データ型のカラムでデフォルト値として使用できます。

INSERT の場合、このデフォルトは CURRENT USER と同じ効果があります。

UPDATE では、LAST USER のデフォルトを持つカラムが明示的に変更されていなければ、現在のユーザー名に変更されます。

DEFAULT TIMESTAMP または DEFAULT UTC TIMESTAMP と結合して使用すると、LAST USER のデフォルトを使用して、ローを最後に変更したユーザーと日時の両方を記録できます (ただし、別々のカラムに記録されます)。

IDENTITY 句　IDENTITY は、DEFAULT AUTOINCREMENT の使用に対する Transact-SQL 互換の代替手段です。SQL Anywhere では、IDENTITY を指定して定義されたカラムは DEFAULT AUTOINCREMENT として実装されます。

column-constraint 句と table-constraint 句　カラム制約とテーブル制約によってデータベース内のデータの整合性が保証されます。整合性制約の違反を起こす文は、実行が完了しません。このような文がエラー検出の前に行った変更は取り消され、エラーがレポートされます。作成できる制約のクラスは、「検査制約」と「参照整合性 (RI) 制約」の 2 つです。検査制約は、データベースに配置されているカラム値が満たす必要がある条件を指定するときに使用されます。RI 制約は、データに一意性の要件を指定するだけでなく、保守が必要な複数テーブルのデータ間に関係を確立します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

RI 制約には、プライマリキー、外部キー、一意性制約という 3 種類があります。RI 制約 (プライマリキー、外部キー、または一意性制約) を作成すると、データベースサーバーは、制約のキーを構築するカラムにインデックスを暗黙的に作成することで、制約を実行します。インデックスは指定した制約のキーに作成されます。キーは、順序付きリストのカラムと各カラムのシーケンス値 (ASC/DESC) で構成されます。

制約はカラムまたはテーブルに指定できます。カラムの制約はテーブル内の 1 つのカラムを指しますが、テーブルの制約はテーブル内の 1 つ以上のカラムを指します。

○ **PRIMARY KEY 句**　プライマリキーはテーブルの各ローをユニークに定義します。プライマリキーは 1 つ以上のカラムで構成されます。1 つのテーブルに複数のプライマリキーが存在することがあります。*column-constraint* 句に PRIMARY KEY を指定すると、そのカラムはテーブルのプライマリキーになります。*table-constraint* で PRIMARY KEY 句を使用して、指定した順序で組み合わせてテーブルのプライマリキーを構築する 1 以上のカラムを指定します。

プライマリキーのカラムの順序は、カラムの順序と一致する必要はありません。つまり、プライマリキーのカラムは、ローの物理的な順序と同じになる必要はありません。また、重複するカラム名を指定することはできません。

プライマリキーを作成すると、キーのインデックスが自動的に作成されます。各カラムに ASC (昇順) または DESC (降順) を指定することで、インデックスの値の順序を指定できます。また、CLUSTERED キーワードを指定して、インデックスをクラスター化するかどうかを指定することもできます。

プライマリキーに含まれるカラムには NULL を使用できません。テーブル内の各ローは、ユニークなプライマリキー値を持ちます。

プライマリキーには、FLOAT や DOUBLE などの概数値データ型を使用しないことをおすすめします。概数値データ型は、算術演算後の丸め誤差がでます。

プライマリキーには空間カラムは設定できません。

○ **FOREIGN KEY 句**　外部キーは、カラムのセットの値がプライマリキーの値、または別のテーブルの一意性制約 (プライマリテーブル) と一致するよう制限します。たとえば、外部キー制約を使って、請求書テーブルの顧客番号が顧客テーブルの顧客番号と確実に一致するようにできます。

外部キーカラムの順序は、テーブルのカラム順を反映する必要はありません。

外部キー指定では、カラム名の重複は許可されません。

UPDATE 操作または DELETE 操作にアクションが指定されていない場合、デフォルトの *action* は RESTRICT になります。

外部キーを作成すると、キーのインデックスが自動的に作成されます。各カラムに ASC (昇順) または DESC (降順) を指定することで、インデックスの値の順序を指定できます。また、CLUSTERED キーワードを指定して、インデックスをクラスター化するかどうかを指定することもできます。

グローバルテンポラリテーブルは、ベーステーブルを参照する外部キーを持つことはできません。また、ベーステーブルも、グローバルテンポラリテーブルを参照する外部キーを持つことはできません。

- **NOT NULL 句**　外部キーカラムを NULL 入力不可にします。外部キー内の NULL は、外部テーブルのこのローに該当するローがプライマリテーブル内にないことを意味します。

- ***role-name* 句**　役割名は外部キーの名前です。役割名の主な機能は、同じテーブルに対する 2 つの外部キーを区別することです。役割名を指定しない場合、役割名は以下のように設定されます。

  1. テーブル名と同じ役割名を含む外部キーが存在しない場合、テーブル名が役割名として割り当てられます。

  2. テーブル名がすでに使用されている場合、役割名は、0 が埋め込まれた、テーブルに固有の 3 桁の数字と結合されたテーブル名になります。

- **REFERENCES 句**　外部キー制約は、REFERENCES カラム制約または FOREIGN KEY テーブル制約を使用して実装できます。このとき、1 つ以上のカラムを指定できます。REFERENCES カラム制約に *column-name* を指定する場合、一意性制約またはプライマリキーの制約を受ける、プライマリテーブルのカラム名を指定します。また、この制約は、その 1 カラムだけで構成します。*column-name* を指定しない場合、外部キーはプライマリテーブルに含まれる単一のプライマリキーのカラムを参照します。

- **MATCH 句**　MATCH 句は、孤立したローの内容と参照整合性違反の内容を管理できるようにすることによって、複数カラムの外部キーを使用した場合に何が一致と見なされるかを決定します。MATCH 句では、また、キーの一意性を指定することで、一意性を別に宣言する必要がなくなります。

  次に、指定できる MATCH タイプを示します。MATCH タイプが一致動作にどのような影響を及ぼすかについては、このトピックの最後にある「例」の項を参照してください。

  - **MATCH [UNIQUE] SIMPLE**　外部キーテーブルのローに一致が発生するのは、すべてのカラム値がプライマリキーテーブルのローにある対応するカラム値と一致する場合です。外部キーテーブルでローが孤立するのは、外部キーの少なくとも 1 つのカラム値が NULL である場合です。

    MATCH SIMPLE はデフォルトの動作です。

    UNIQUE キーワードを指定すると、NULL 以外のキー値に対して、参照テーブルで一致が 1 つのみになります。

  - **MATCH [UNIQUE] FULL**　外部キーテーブルのローに一致が発生するのは、いずれの値も NULL ではなく、値がプライマリキーテーブルのローにある対応するカラム値と一致する場合です。ローが孤立するのは、外部キーのすべてのカラム値が NULL の場合です。

    UNIQUE キーワードを指定すると、NULL 以外のキー値に対して、参照テーブルで一致が 1 つのみになります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **UNIQUE 句**　*column-constraint* 句では、UNIQUE 制約はカラム値をユニークにする必要があることを示します。*table-constraint* 句では、UNIQUE 制約は、テーブル内の各ローをユニークに識別する 1 つ以上のカラムを指定します。テーブル内の異なるローが、指定されているすべてのカラムで同じ値を持つことはできません。1 つのテーブルは複数の UNIQUE 制約を持つことができます。

UNIQUE 制約はユニークインデックスとは異なります。ユニークインデックスのカラムは NULL でもかまいません。一方、UNIQUE 制約のカラムは NULL にはなりません。また、外部キーは、プライマリキーまたは UNIQUE 制約を参照できますが、ユニークインデックスは参照できません。ユニークインデックスは NULL の複数のインスタンスを含むことがあるからです。

UNIQUE 制約のカラムは、任意の順序で指定できます。また、各カラムに ASC (昇順) または DESC (降順) を指定することで、自動的に作成された対応するインデックスの値の順序を指定できます。ただし、カラム名を重複して指定することはできません。

一意性制約には、カラムに FLOAT や DOUBLE などの概数値データ型を使用しないことをおすすめします。概数値データ型は、算術演算後の丸め誤差がでます。

また、CLUSTERED キーワードを指定して、制約をクラスター化するかどうかを指定することもできます。

○ **CHECK 句**　この制約で、任意の条件を検証できます。たとえば、CHECK 制約を使うと、Sex というカラムには M または F の値だけが確実に入ります。

テーブル内の 2 つ以上のカラム間の関係 (カラム A はカラム B 未満である必要があるなど) が関与する CHECK 制約の作成が必要な場合は、代わりにテーブル制約を定義します。

テーブルのどのローも CHECK 制約に違反することはできません。INSERT または UPDATE 文によってローが制約に違反する場合、操作は許可されず、この文の処理結果は取り消されます。変更は、CHECK 制約条件の評価結果が FALSE の場合にのみ拒否されます。CHECK 制約条件の評価結果が TRUE または UNKNOWN の場合、変更は許可されます。

○ **COMPUTE 句**　COMPUTE 句は、*column-constraint* 句にのみ使用します。

カラムが COMPUTE 句を使って作成される場合、すべてのローの値は式で提供されます。この制約を付けて作成されたカラムは、読み込み専用カラムです。この値は、ローが変更されるたびにデータベースサーバーによって変更されます。COMPUTE 式は、非決定的な値を返しません。たとえば、CURRENT TIMESTAMP などの特別値や非決定的関数は指定できません。COMPUTE 式が非決定的な値を返したとしても、それをクエリに含まれる式との一致に使用することはできません。

リモートテーブルでは COMPUTE 句は無視されます。

計算カラムの値を変更しようとする UPDATE 文は、そのカラムに対応するトリガーを起動します。

CHECK ON COMMIT 句　CHECKONCOMMIT オプションは wait_for_commit データベースオプションを上書きします。この句を指定すると、データベースサーバーは COMMIT を待ってから外部キーに対する RESTRICT アクションをチェックします。CHECK ON COMMIT オプション

を指定することによって、外部キーチェックの遅延が発生しますが、CASCADE、SET NULL、SET DEFAULT、検査制約などの他のアクションの遅延は発生しません。

FOR OLAP WORKLOAD 句　外部キー定義の REFERENCES 句に FOR OLAP WORKLOAD を指定すると、データベースサーバーは特定の最適化を実行し、キーに関する統計情報を収集して、OLAP の負荷に関するパフォーマンスを向上させることができます。特に、optimization_workload を OLAP に設定すると有効です。

PCTFREE 句　各テーブルページに確保する空き領域の割合を指定します。空き領域は、データが更新されたときにローのサイズが増えた場合に使用されます。テーブルページに空き領域がない場合は、ページのローのサイズが増えるたびに、ローを複数のテーブルページに分割することが必要になり、ローの断片化が発生します。また、パフォーマンス低下の可能性があります。

percent-free-space の値は、0 ～ 100 までの整数です。0 を指定すると、各ページに空き領域を残さず、各ページを完全にパックします。高い値に設定すると、各ローは単独でページに挿入されます。PCTFREE が設定されない場合、または削除された場合、データベースのページサイズに応じたデフォルトの PCTFREE 値が適用されます (ページサイズが 4 KB 以上の場合は 200 バイト)。PCTFREE の値は、ISYSTAB システムテーブルに格納されます。

備考

CREATE TABLE 文は新しいテーブルを作成します。所有者名を指定することにより、別のユーザーに対するテーブルを作成できます。GLOBAL TEMPORARY を指定すると、テーブルはテンポラリテーブルと見なされます。指定しないと、テーブルはベーステーブルとなります。

CREATE TABLE 文のテーブル名の前にシャープ記号 (#) を付けてテーブルを作成すると、テンポラリテーブルが宣言されます。テンポラリテーブルは現在の接続でのみ使用できます。シャープ記号 (#) を使用して作成されたテンポラリテーブルは、ON COMMIT PRESERVE ROWS 句を使用して作成されたテーブルと同じです。

同じスコープ内にある 2 つのローカルテンポラリテーブルは、同じ名前にはできません。ベーステーブルと同じ名前のテンポラリテーブルを作成すると、ローカルテンポラリテーブルのスコープが終了した時点で、そのベーステーブルは、その接続内でのみ参照できるようになります。接続では、既存のテンポラリテーブルと同じ名前のベーステーブルを作成できません。

SQL Anywhere のカラムはデフォルトで NULL を許容しています。この設定は、allow_nulls_by_default データベースオプションを使用して制御できます。

パーミッション

DBA 権限または RESOURCE 権限

関連する動作

オートコミット (グローバルテンポラリテーブルを作成しているときにも適用)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「OLAP のサポート」『SQL Anywhere サーバー SQL の使用法』
- 「allow_nulls_by_default オプション」『SQL Anywhere サーバー データベース管理』
- 「イベント」『SQL Anywhere サーバー データベース管理』
- 「sa_reset_identity システムプロシージャー」1162 ページ
- 「シーケンス値と AUTOINCREMENT 値からの選択」『SQL Anywhere サーバー SQL の使用法』
- 「AUTOINCREMENT カラムのあるテーブルの再ロード」『SQL Anywhere 12 変更点とアップグレード』
- 「CREATE LOCAL TEMPORARY TABLE 文」571 ページ
- 「@@identity グローバル変数」83 ページ
- 「特殊な IDENTITY カラム」『SQL Anywhere サーバー SQL の使用法』
- 「データベース内のテーブル暗号化の有効化」『SQL Anywhere サーバー データベース管理』
- 「ALTER TABLE 文」462 ページ
- 「CREATE DBSPACE 文」528 ページ
- 「CREATE EXISTING TABLE 文」546 ページ
- 「DECLARE LOCAL TEMPORARY TABLE 文」691 ページ
- 「DROP TABLE 文」731 ページ
- 「optimization_workload オプション」『SQL Anywhere サーバー データベース管理』
- 「特別値」61 ページ
- 「SQL データ型」85 ページ
- 「テーブルの作成」『SQL Anywhere サーバー SQL の使用法』
- 「allow_nulls_by_default オプション」『SQL Anywhere サーバー データベース管理』
- 「テンポラリテーブル」『SQL Anywhere サーバー SQL の使用法』
- 「計算カラム」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE DBSPACE 文」
- 「追加の DB 領域の考慮事項」『SQL Anywhere サーバー データベース管理』
- 「ユニークな値を生成するためのシーケンスの使用」『SQL Anywhere サーバー SQL の使用法』
- 「default_dbspace オプション」『SQL Anywhere サーバー データベース管理』
- 「DECLARE LOCAL TEMPORARY TABLE 文」691 ページ

標準と互換性

- **SQL/2008**　CREATE TABLE は SQL/2008 標準のコア機能ですが、SQL Anywhere でサポートされている一部のコンポーネントはオプションの SQL 言語機能です。これらの機能のサブセットを次に示します。

  ○テンポラリテーブルのサポートは、SQL 言語機能 F531 です。

  ○IDENTITY カラムのサポートは SQL 機能 T174 ですが、SQL Anywhere では標準とは少し異なる構文が使用されます。

  ○外部キー制約のサポートには、SQL 言語機能 T191 "Referential action: RESTRICT"、F741 "Referential MATCH types"、F191 "Referential delete actions"、F701 "Referential update actions" が含まれています。SQL Anywhere では MATCH PARTIAL がサポートされていないことに注意してください。

  SQL Anywhere では、SQL 言語機能 T591 (UNIQUE constraints of possibly null columns) はサポートされていません。SQL Anywhere では、PRIMARY KEY 制約または UNIQUE 制約の一部であるすべてのカラムを NOT NULL と宣言してください。

CREATE TABLE の次のコンポーネントは、ベンダー拡張です。

○{ IN | ON } *dbspace-name* 句

○ENCRYPTED 句、NOT TRANSACTIONAL 句、SHARE BY ALL 句

○カラム定義の COMPRESSED, INLINE 句、PREFIX 句、NO INDEX 句

○AUTOINCREMENT、GLOBAL AUTOINCREMENT、CURRENT DATABASE、CURRENT REMOTE USER、CURRENT UTC TIMESTAMP、ほとんどの特別値などの、実装依存のさまざまな DEFAULT 値。また、SEQUENCE ジェネレーターを参照する DEFAULT 句も、ベンダー拡張です。

○MATCH UNIQUE の指定

○PRIMARY KEY 句または FOREIGN KEY 句での指定値のソート方法 (ASC または DESC)

○参照先テーブルの PRIMARY KEY 句での指定と異なる順序で FOREIGN KEY カラムを指定する機能

例

次の例は、図書データベース用にテーブルを作成し、図書情報を保持します。

```
CREATE TABLE library_books (
  -- NOT NULL is assumed for primary key columns
  isbn CHAR(20) PRIMARY KEY,
  copyright_date DATE,
  title CHAR(100),
  author CHAR(50),
  -- column(s) corresponding to primary key of room
  -- are created automatically
  FOREIGN KEY location REFERENCES room
);
```

次の例では、図書データベース用にテーブルを作成して、貸し出し図書の情報を保持します。date_borrowed のデフォルト値は、エントリが作成される日に本が貸し出されることを示します。date_returned カラムは、本が返却されるまでは NULL です。

```
CREATE TABLE borrowed_book (
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  date_returned DATE,
  book CHAR(20)
  REFERENCES library_books (isbn),
  -- The check condition is UNKNOWN until
  -- the book is returned, which is allowed
CHECK( date_returned >= date_borrowed )
);
```

次の例は、販売データベース用にテーブルを作成し、注文と注文項目情報を保持します。

```
CREATE TABLE Orders (
  order_num INTEGER NOT NULL PRIMARY KEY,
  date_ordered DATE,
  name CHAR(80)
);
CREATE TABLE Order_item (
```

```
    order_num INTEGER NOT NULL,
    item_num SMALLINT NOT NULL,
    PRIMARY KEY ( order_num, item_num ),
    -- When an order is deleted, delete all of its
    -- items.
    FOREIGN KEY ( order_num )
    REFERENCES Orders ( order_num )
    ON DELETE CASCADE
);
```

次の例は、架空のリモートサーバー SERVER_A にテーブル t1 を作成し、このリモートテーブルにマッピングするプロキシテーブル t1 を作成します。

```
CREATE TABLE t1
( a INT,
  b CHAR(10) )
AT 'SERVER_A.db1.joe.t1';
```

次の例は、2 つのテーブル Table1 と Table2 を作成し、外部キーを Table2 に追加し、値を Table1 に挿入します。最後の文は、値を Table2 に挿入しようとします。挿入しようとした値は Table1 と単純一致しないため、エラーが返されます。

```
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 );
```

次の文は、キーの一部 (すべてではない) のカラムが NULL の場合に、MATCH SIMPLE と MATCH SIMPLE UNIQUE で、複数カラムの外部キーの処理方法がどのように異なるかを示します。

```
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); //This statement will insert 2 rows.

CREATE TABLE ft2( fpk INT PRIMARY KEY, FOREIGN KEY (ref) REFERENCES pt MATCH UNIQUE
SIMPLE);
INSERT INTO ft2 VALUES(100,1), (200,1);「//This statement will fail because the values for the second
column are not unique.」
```

次の文は、MATCH SIMPLE と MATCH UNIQUE SIMPLE がどのように異なるかを示します。

```
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;

INSERT INTO ft3 VALUES(100,1,10);
// MATCH SIMPLE test succeeds; all column values match the corresponding values in pt2.

INSERT INTO ft3 VALUES(200,null,null);
// MATCH SIMPLE test succeeds; at least one column in the key is null.

INSERT INTO ft3 VALUES(300,2,null);
// MATCH SIMPLE test succeeds; at least one column in the key is null.

INSERT INTO ft4 VALUES(100,1,10);
// MATCH FULL test succeeds; all column values match the corresponding values in pt2.

INSERT INTO ft4 VALUES(200,null,null);
// MATCH FULL test succeeds; all column values in the key are null.

INSERT INTO ft4 VALUES(300,2,null);
// MATCH FULL test fails; both columns in the key must be null or match the corresponding values in
pt2.
```

CREATE TEXT CONFIGURATION 文

テキストインデックスの構築と更新に使用するテキスト設定オブジェクトを作成します。

構文

CREATE TEXT CONFIGURATION [*owner.*]*new-config-name*
FROM [*owner.*]*existing-config-name*

パラメーター

FROM 句　新しいテキスト設定オブジェクトを作成するときのテンプレートとして使用するテキスト設定オブジェクトの名前を指定します。デフォルトのテキスト設定オブジェクトの名前は、default_char と default_nchar です。

テキスト設定オブジェクトを作成すると、日付値と時刻値の文字列への変換方法に影響を及ぼすデータベースオプションが、default_char と default_nchar のテキスト設定オブジェクトテンプレートからコピーされます。

備考

別のテキスト設定オブジェクトをテンプレートとして使用してテキスト設定オブジェクトを作成し、その後、必要に応じて ALTER TEXT CONFIGURATION 文を使用してオプションを変更します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

データベース内のすべてのテキスト設定オブジェクトのリストとその設定を表示するには、SYSTEXTCONFIG システムビューを問い合わせます。

パーミッション

DBA 権限または RESOURCE 権限が必要です。

すべてのテキスト設定オブジェクトには PUBLIC アクセスがあります。テキストインデックスを作成できるパーミッションを持つユーザーであれば、任意のテキスト設定オブジェクトを使用できます。

関連する動作

オートコミット。

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「デフォルトのテキスト設定オブジェクト」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトに影響するデータベースオプション」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの作成または変更時の指定項目」『SQL Anywhere サーバー SQL の使用法』
- 「SYSTEXTCONFIG システムビュー」1320 ページ
- 「チュートリアル：GENERIC テキストインデックスへの全文検索の実行」『SQL Anywhere サーバー SQL の使用法』
- 「チュートリアル：あいまい全文検索の実行」『SQL Anywhere サーバー SQL の使用法』
- 「ALTER TEXT CONFIGURATION 文」476 ページ
- 「DROP TEXT CONFIGURATION 文」732 ページ
- 「sa_refresh_text_indexes システムプロシージャー」1158 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の CREATE TEXT CONFIGURATION 文は、default_char テキスト設定オブジェクトを使用して、テキスト設定オブジェクト max_term_sixteen を作成します。後続の ALTER TEXT CONFIGURATION 文は、max_term_sixteen の最大単語長を 16 に変更します。

```
CREATE TEXT CONFIGURATION max_term_sixteen FROM default_char;
ALTER TEXT CONFIGURATION max_term_sixteen
  MAXIMUM TERM LENGTH 16;
```

CREATE TEXT INDEX 文

テキストインデックスを作成します。

構文

CREATE TEXT INDEX [**IF NOT EXISTS**] *text-index-name*
ON [*owner.*]*table-name*(*column-name*, ...)

[**IN** *dbspace-name*]
[**CONFIGURATION** [*owner*.]*text-configuration-name*]
[{ **IMMEDIATE REFRESH**
| **MANUAL REFRESH**
| **AUTO REFRESH** [**EVERY** *integer* { **MINUTES** | **HOURS** }]]
}

パラメーター

IF NOT EXISTS 句　IF NOT EXISTS 句が指定され、指定されたテキストインデックスがすでに存在する場合は、変更は行われず、エラーも返されません。

ON 句　この句は、テキストインデックスを構築するテーブルとカラムを指定するときに使用します。

IN 句　この句は、テキストインデックスが格納される DB 領域を指定するときに使用します。この句が指定されていない場合は、テキストインデックスが参照するテーブルと同じ DB 領域でテキストインデックスが作成されます。

CONFIGURATION 句　この句は、テキストインデックスの作成時に使用するテキスト設定オブジェクトを指定するときに使用します。この句が指定されておらず、インデックスのカラムのいずれかが NCHAR の場合は、default_nchar テキスト設定オブジェクトが使用されます。それ以外の場合は、default_char テキスト設定オブジェクトが使用されます。

REFRESH 句　この句は、テキストインデックスの再表示タイプを指定するときに使用します。REFRESH 句を指定しない場合、デフォルトとして IMMEDIATE REFRESH が使用されます。次に、指定できる再表示タイプを示します。

- **IMMEDIATE REFRESH**　IMMEDIATE REFRESH は、基本となるテーブルの変更によってテキストインデックスのデータが影響を受けるたびにテキストインデックスを再表示する場合に指定します。

- **AUTO REFRESH**　この句は、内部サーバーイベントを使用して自動的にマテリアライズドビューをリフレッシュするときに使用します。EVERY 句は、再表示間隔を分または時間単位で指定するときに使用します。間隔情報を指定しないで AUTO REFRESH を指定した場合、データベースサーバーは 60 分ごとにテキストインデックスを再表示します。
  sa_text_index_stats システムプロシージャーから返された pending_size value が、最後のリフレッシュ時にテキストインデックスサイズの 20 % を超えるか、deleted_length がテキストインデックスサイズの 50 % を超えると、AUTO REFRESH 句で指定された時間の前にテキストインデックスがリフレッシュされる場合があります。1 分ごとに内部イベントが実行され、AUTO REFRESH の全テキストインデックスに対してこの条件がチェックされます。

- **MANUAL REFRESH**　この句は、テキストインデックスが手動で再表示されるように指定するときに使用します。

再表示タイプの詳細については、「テキストインデックスの再表示タイプ」『SQL Anywhere サーバー SQL の使用法』を参照してください。

備考

テキストインデックスは、ビュー、マテリアライズドビュー、またはテンポラリテーブルには作成できません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

IMMEDIATE REFRESH のテキストインデックスは、作成時に移植され、この初期再表示中に、テーブルに排他ロックがかけられます。IMMEDIATE REFRESH のテキストインデックスは、スナップショットアイソレーションを使用するクエリを完全にサポートします。

MANUAL と AUTO REFRESH のテキストインデックスは、作成後に初期化 (再表示) されます。

AUTO REFRESH のテキストインデックスの再表示では、独立性レベル 0 を使用してテーブルがスキャンされます。

テキストインデックスの作成後に、その定義を IMMEDIATE REFRESH に変更したり、IMMEDIATE REFRESH から変更したりできません。いずれかの変更が必要な場合は、テキストインデックスを削除して再度作成する必要があります。

REFRESH TEXT INDEX 文を使用して、AUTO REFRESH テキストインデックスを手動で再表示することを選択できます。

テキストインデックスとそれらが参照するテキスト設定オブジェクトを表示する方法については、「データベース内のテキストインデックスに関する情報の表示 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』を参照してください。

パーミッション

基本となるテーブルの所有者であるか、DBA 権限または REFERENCES パーミッションが必要です。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、この文は実行できません。

関連する動作

オートコミット。

参照

- 「チュートリアル：GENERIC テキストインデックスへの全文検索の実行」『SQL Anywhere サーバー SQL の使用法』
- 「isolation_level オプション」『SQL Anywhere サーバー データベース管理』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』
- 「REFRESH TEXT INDEX 文」870 ページ
- 「チュートリアル：あいまい全文検索の実行」『SQL Anywhere サーバー SQL の使用法』
- 「テキストインデックスの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「SYSTEXTCONFIG システムビュー」1320 ページ
- 「ALTER TEXT INDEX 文」480 ページ
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「sa_char_terms システムプロシージャー」1043 ページ
- 「sa_nchar_terms システムプロシージャー」1139 ページ
- 「sa_refresh_text_indexes システムプロシージャー」1158 ページ
- 「sa_text_index_stats システムプロシージャー」1201 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、サンプルデータベースの MarketingInformation テーブルの Description カラムについてテキストインデックス myTxtIdx を作成します。MarketingTextConfig テキスト設定オブジェクトが使用され、再表示間隔は 24 時間ごとに設定されます。

```
CREATE TEXT INDEX myTxtIdx ON MarketingInformation ( Description )
  CONFIGURATION default_char
  AUTO REFRESH EVERY 24 HOURS;
```

CREATE TRIGGER 文

テーブル内にトリガーを作成します。

構文

CREATE [**OR REPLACE**] **TRIGGER** *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 : BEGIN 文。「BEGIN 文」497 ページを参照してください。

パラメーター

OR REPLACE 句　OR REPLACE を指定すると、新しいトリガーが作成されるか、同じ名前の既存のトリガーが置き換えられます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

trigger-event トリガーは次のイベントによって起動できます。DELETE、INSERT、または UPDATE イベントに複数のトリガーを定義するか、UPDATE OF *column-list* イベントに 1 つのトリガーを定義できます。

○ **DELETE 句** 関連するテーブルのローが削除されると、呼び出されます。

○ **INSERT 句** このトリガーに関連するテーブルに新しいローが挿入されると、呼び出されます。

○ **UPDATE 句** 関連するテーブルのローが更新されると、呼び出されます。

○ **UPDATE OF *column-list* 句** 関連するテーブルのローが更新され、*column-list* のカラムが修正されると、呼び出されます。このタイプのトリガーイベントは、*trigger-event-list* では使用できません。このトリガー用に定義されたトリガーイベントであることが必要です。この句は、INSTEAD OF トリガーでは使用できません。

処理が必要なイベントごとにトリガーを個別に作成できます。または、共有するアクションや、イベントに応じたアクションが複数ある場合は、すべてのイベントに対して 1 つのトリガーを作成し、IF 文を使用して実行するアクションを区別できます。

trigger-type ローレベルのトリガーを定義して、挿入、更新、または削除の前 (BEFORE)、後 (AFTER) または代わり (INSTEAD OF) に実行できます。文レベルのトリガーは、文の INSTEAD OF または AFTER の実行を定義できます。

BEFORE UPDATE トリガーは、ローを対象に UPDATE が実行されるたび、新しい値が古い値と異なるかどうかに関係なく起動されます。つまり、BEFORE UPDATE トリガーに *column-list* が指定されている場合は、UPDATE 文の SET 句に *column-list* 内のカラムがあると、トリガーが起動します。AFTER UPDATE トリガーに *column-list* が指定されている場合は、*column-list* 内のカラムの値が UPDATE 文によって**変更**されると、トリガーが起動します。

INSTEAD OF トリガーは、通常のビューに定義できるトリガーの唯一の形式です。INSTEAD OF トリガーは、他の動作を、トリガー動作で置換します。INSTEAD OF トリガーが起動すると、トリガー動作はスキップされ、指定された動作が実行されます。INSTEAD OF トリガーは、ローレベルまたは文レベルで定義できます。文レベルの INSTEAD OF トリガーは、ローレベルのすべての処理を含め、文全体を置換します。文レベルの INSTEAD OF トリガーが起動すると、その文の結果としてローレベルのトリガーが起動することはありません。ただし、文レベルのトリガーの本文が、その他の処理を実行するため、結果として、その他のローレベルのトリガーが実行されます。

INSTEAD OF トリガーを定義する場合は、UPDATE OF *column-list* 句、ORDER 句、または WHEN 句は使用できません。

RESOLVE トリガー型は、SQL Remote とともに使用します。これは、ローレベルの UPDATE または UPDATE OF *column-list* の前だけで起動されます。

ORDER 句 同じタイミング (before、after、resolve) で起動する同じタイプ (insert、update、delete) の追加トリガーを定義する場合は、ORDER 句を指定してトリガーを起動する順序をデータベースサーバーに指示します。同じタイミングで起動するように設定された同じタイプのトリガーの間では、順序番号をユニークにします。ユニークでない順序番号を指定すると、エラーが返さ

れます。順序番号は、連続した順番にする必要はありません (たとえば、1、12、30 と指定できます)。データベースサーバーは、最下位の番号が付いたトリガーから順に起動します。

ORDER 句を省略するか、0 を指定すると、データベースサーバーは順序番号 1 を割り当てます。ただし、すでに同じタイプの別のトリガーが 1 に設定されている場合は、エラーが返されます。

追加トリガーを追加する場合は、トリガーの動作が相互に影響するかどうかによって、イベントの同じタイプの既存トリガーを修正する必要が生じる可能性があります。相互に影響しない場合、新しいトリガーには既存のトリガーより大きな ORDER 値が必要です。相互に影響する場合は、他のトリガーの動作を検討する必要があり、これらのトリガーが起動する順序の変更が必要となる可能性があります。

ORDER 句は、INSTEAD OF トリガーではサポートされません。テーブルまたはビューに定義されたタイプ (insert、update、delete) ごとに、1 つの INSTEAD OF トリガーだけを使用できます。

REFERENCING 句　REFERENCING OLD 句と REFERENCING NEW 句を使用すると、挿入、削除、または更新されたローを参照できます。この句では、UPDATE は削除とそれに続く挿入として取り扱われます。

INSERT は REFERENCING NEW 句を使います。これは挿入されたローを表します。REFERENCING OLD 句はありません。

DELETE は REFERENCING OLD 句を使います。これは削除されたローを表します。REFERENCING NEW 句はありません。

UPDATE は、REFERENCING OLD 句を使うときは更新前のローを表し、REFERENCING NEW 句を使うときは更新後のローを表します。

REFERENCING OLD と REFERENCING NEW の意味は、トリガーがローレベルのトリガーと文レベルのトリガーのどちらかによって異なります。ローレベルのトリガーの場合、REFERENCING OLD 句を使うと、更新または削除する前のローの値を参照できます。また、REFERENCING NEW 句を使用すると、挿入または更新された値を参照できます。OLD ローと NEW ローは、BEFORE と AFTER トリガーの中で参照できます。REFERENCING NEW 句を使うと、BEFORE トリガーの新しいローを修正してから、挿入または更新操作を行うことができます。

文レベルのトリガーの場合、REFERENCING OLD と REFERENCING NEW 句は、ローの古い値と新しい値を保持している宣言されたテンポラリテーブルを参照します。

REFERENCING REMOTE 句は SQL Remote で使用します。これを使うと、UPDATE 文の VERIFY 句に設定されている値を参照できます。これは、必ずカラムリストのトリガー RESOLVE UPDATE または RESOLVE UPDATE OF と一緒に使用してください。

FOR EACH 句　トリガーをローレベルのトリガーとして宣言するには、FOR EACH ROW 句を使用します。トリガーを文レベルのトリガーとして宣言するには、FOR EACH STATEMENT 句を使用するか、または FOR EACH 句を省略します。文レベルのトリガーを宣言する場合は、わかりやすくするために、FOR EACH STATEMENT 句を指定することをおすすめします。

WHEN 句　探索条件が真と評価されたローに対してだけ、トリガーが起動されます。WHEN 句は、ローレベルトリガーと一緒にだけ使用できます。この句は、INSTEAD OF トリガーでは使用できません。

trigger-body トリガーの本文には、トリガーアクションが生じたときに実行されるアクションが含まれます。これは、BEGIN 文で構成されます。

BEGIN 文にトリガーオペレーション条件を指定できます。トリガーオペレーション条件は、トリガーを起動したトリガーイベントに従って動作を行います。たとえば、トリガーが更新と削除の両方のために起動するように定義されている場合は、2 つの条件に異なる動作を指定できます。

備考

CREATE TRIGGER 文は、データベースのテーブルに関連するトリガーを作成し、データベースにトリガーを格納します。

マテリアライズドビューにトリガーを定義することはできません。定義すると、SQLE_INVALID_TRIGGER_MATVIEW エラーが返されます。

トリガーをローレベルのトリガーとして宣言すると、各ローを修正する前または後にトリガーが実行されます。また、トリガーを文レベルのトリガーとして宣言すると、トリガー文全体が完了してから、トリガーが実行されます。

パーミッション

RESOURCE 権限またはテーブルに対する ALTER パーミッションが必要です。または、テーブルの所有者であるか、DBA 権限が必要です。CREATE TRIGGER はテーブルにテーブルロックを設定して、テーブルを排他的に使用します。

関連する動作

オートコミット。

参照

- 「BEGIN 文」497 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「CREATE TRIGGER 文 [T-SQL]」676 ページ
- 「DROP TRIGGER 文」734 ページ
- 「ROLLBACK TRIGGER 文」893 ページ
- 「UPDATE 文」976 ページ
- 「ALTER TRIGGER 文」481 ページ
- 「RAISERROR 文」862 ページ
- 「CONFLICT 関数 [その他]」173 ページ
- 「INSTEAD OF トリガー」『SQL Anywhere サーバー SQL の使用法』
- 「整合性制約を実装するための SQL 文」『SQL Anywhere サーバー SQL の使用法』
- 「トリガーの作成」『SQL Anywhere サーバー SQL の使用法』
- 「トリガー」『SQL Anywhere サーバー SQL の使用法』
- 「トリガーの実行」『SQL Anywhere サーバー SQL の使用法』
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』
- 「トリガーオペレーション条件」58 ページ
- 「トリガーを使用したカスタム競合解決」『SQL Remote』

標準と互換性

- **SQL/2008**　CREATE TRIGGER は、SQL/2008 標準のオプションの SQL 言語機能 T211、"Basic trigger capability" の一部です。ROW トリガーはオプションの SQL 言語機能 T212 であり、INSTEAD OF トリガーはオプションの SQL 言語機能 T213 です。

  SQL Anywhere トリガーの一部の機能は、ベンダー拡張です。これらのコードを以下に示します。

  ○オプションの OR REPLACE 構文。既存のトリガーを置き換える場合、新しいトリガーインスタンスの作成の認証は省略されます。

  ○ORDER 句。SQL/2008 では、トリガーは作成順に起動されます。

  ○RESOLVE トリガーはベンダー拡張です。

- **Transact-SQL**　ROW トリガーと RESOLVE トリガーは、Adaptive Server Enterprise でサポートされていません。Transact-SQL の INSTEAD OF トリガーは、Adaptive Server Enterprise ではサポートされていますが、SQL Anywhere の Transact-SQL ダイアレクトではサポートされていません。Transact-SQL トリガーは、異なる構文で定義されます。

例

この例は、文レベルのトリガーを作成します。まず、テーブルを作成する必要があります。

```
CREATE TABLE t0
( id integer NOT NULL,
 times timestamp NULL DEFAULT current timestamp,
 remarks text NULL,
 PRIMARY KEY ( id )
);
```

次に、このテーブルについて文レベルのトリガーを作成します。

```
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;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

次の例は、前の例で作成された myTrig トリガーを置き換えます。

```
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;
```

次の例は、BEFORE UPDATE トリガーで REFERENCING NEW を使用する方法を示します。次の例では、新規 Employees テーブルの郵便番号が大文字になるようにしています。

```
CREATE TRIGGER emp_upper_postal_code
BEFORE UPDATE OF PostalCode
ON 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 Employees SET state='ON', PostalCode='n2x 4y7' WHERE EmployeeID=191;
SELECT PostalCode FROM Employees WHERE EmployeeID = 191;
```

次の例は、BEFORE DELETE トリガーで REFERENCING OLD を使用する方法を示します。この例は、Employees テーブルから、退職していない従業員が削除されないようにします。

```
CREATE TRIGGER TR_check_delete_employee
BEFORE DELETE
ON Employees
REFERENCING OLD AS current_employees
FOR EACH ROW /* WHEN( search_condition ) */
BEGIN
  IF current_employees.TerminationDate IS NULL THEN
   RAISERROR 30001 'You cannot delete an employee who has not been fired';
   END IF;
END;
```

次の例は、BEFORE UPDATE トリガーで REFERENCING NEW と REFERENCING OLD を使用する方法を示します。この例は、従業員の給料に減給が生じないようにします。

```
CREATE TRIGGER TR_check_salary_decrease
   BEFORE UPDATE
     ON 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;
```

次の例は、BEFORE UPDATE トリガーで REFERENCING NEW と REFERENCING OLD を使用する方法を示します。この例でも、従業員の給料に減給が生じないようにしますが、このトリガーは給料のカラムが更新されたときだけ起動するため、より効率的になっています。

```
CREATE TRIGGER TR_check_salary_decrease_column
   BEFORE UPDATE OF Salary
     ON Employees
   REFERENCING OLD AS before_update
   NEW AS after_update
FOR EACH ROW /* WHEN( search_condition ) */
BEGIN
  IF after_update.salary < before_update.salary THEN
   RAISERROR 30002 'You cannot decrease a salary';
End IF;
END;
```

次の例は、BEFORE INSERT トリガーと UPDATE トリガーで REFERENCING NEW を使用する方法を示します。この例では、SalesOrderItems テーブルのローで挿入や更新が行われる前に起動する、トリガーを作成します。

```
CREATE TRIGGER TR_update_date
  BEFORE INSERT, UPDATE
    ON SalesOrderItems
  REFERENCING NEW AS new_row
FOR EACH ROW
BEGIN
  SET new_row.ShipDate = CURRENT TIMESTAMP;
END;
```

CREATE TRIGGER 文 [T-SQL]

Adaptive Server Enterprise 互換の方法で、データベース内に新しいトリガーを作成します。

構文 1

CREATE TRIGGER [*owner*.]*trigger_name*
ON [*owner*.]*table_name*
FOR { **INSERT**, **UPDATE**, **DELETE** }
AS *statement-list*

構文 2

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

備考

CREATE TRIGGER は、テーブルに対する排他的なテーブルロックを取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

削除または挿入されたローは、2 つの宣言されたテンポラリテーブル内に保持されます。Transact-SQL トリガーでは、Adaptive Server Enterprise の場合と同様に、テーブル名 "deleted" と "inserted" を使用してこれらのローにアクセスできます。Watcom-SQL CREATE TRIGGER 文では、REFERENCING 句を使用してこれらのローを参照します。

トリガー名はデータベース内でユニークでなければなりません。

Transact-SQL トリガーは、そのトリガーを起動した文が実行された後に実行されます。

Transact-SQL トリガーの作成時には ORDER 句がサポートされていないため、trigger_order の値は 1 に設定されます。SYSTRIGGER システムテーブルには、table_id、event、trigger_time、trigger_order というユニークなインデックスがあります。特定のイベント (挿入、更新、削除) では、文レベルのトリガーは常に AFTER であり、trigger_order は設定できません。このため、その他のトリガーで 1 以外の順序に設定されないことを前提として、テーブルごとに各タイプのいずれか 1 つのみを設定できます。

パーミッション

テーブルに対する DBA 権限、または RESOURCE 権限と ALTER パーミッション。

関連する動作

オートコミット。

参照

- 「CREATE TRIGGER 文」670 ページ
- 「Transact-SQL のトリガー」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　ROW トリガーは、Adaptive Server Enterprise でサポートされていません。Transact-SQL の INSTEAD OF トリガーは、Adaptive Server Enterprise ではサポートされていますが、SQL Anywhere の Transact-SQL ダイアレクトではサポートされていません。

CREATE USER 文

データベースユーザーまたはグループを作成します。

構文

CREATE USER *user-name* [**IDENTIFIED BY** *password*]
[**LOGIN POLICY** *policy-name*]
[**FORCE PASSWORD CHANGE** { **ON** | **OFF** }]

パラメーター

user-name　作成するユーザーの名前。

IDENTIFIED BY 句　ユーザーのパスワード。パスワードのないユーザーは、データベースに接続できません。

policy-name ユーザーに割り当てるログインポリシーの名前。ログインポリシーが指定されない場合、DEFAULT ログインポリシーが適用されます。

FORCE PASSWORD CHANGE 句 ログイン時にユーザーが新しいパスワードを指定する必要があるかどうかを制御します。この設定は、ユーザーのポリシーの password_expiry_on_next_login オプション設定を上書きします。

備考

ユーザーにパスワードを指定する必要はありません。パスワードのないユーザーは、データベースに接続できません。これは、グループを作成し、他のユーザーはグループユーザー ID を使用してデータベースに接続させないようにする場合に便利です。ユーザー ID には、有効な ID を使用します。

●ユーザー ID には、次に該当する値を指定できません。

○ 空白スペース、一重引用符、または二重引用符で始まる値
○ 空白スペースで終わる値
○ セミコロンを含む値

●パスワードは大文字と小文字が区別され、次に該当する値は指定できません。

○ 空白スペース、一重引用符、または二重引用符で始まる値
○ 空白スペースで終わる値
○ セミコロンを含む値
○ 256 バイト長を超える値

パーミッション

DBA 権限

関連する動作

なし

参照

- 「CREATE LOGIN POLICY 文」573 ページ
- 「ALTER LOGIN POLICY 文」424 ページ
- 「ALTER USER 文」482 ページ
- 「COMMENT 文」512 ページ
- 「DROP LOGIN POLICY 文」716 ページ
- 「DROP USER 文」735 ページ
- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』
- 「ユーザーの作成とログインポリシーの割り当て」『SQL Anywhere サーバー データベース管理』
- 「GRANT 文」782 ページ
- 「min_password_length オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、ユーザー SQLTester を作成し、パスワードを welcome に設定します。SQLTester ユーザーには Test1 ログインポリシーが割り当てられ、次回のログイン時にパスワードの有効期限が切れます。

```
CREATE USER SQLTester IDENTIFIED BY welcome
LOGIN POLICY Test1
FORCE PASSWORD CHANGE ON;
```

次の例は、グループ MyGroup を作成します。

```
CREATE USER MyGroup;
GRANT GROUP TO MyGroup;
```

CREATE VARIABLE 文

SQL 変数を作成します。

構文

CREATE [**OR REPLACE**] **VARIABLE** *identifier data-type* [{ **=** | **DEFAULT** } *initial-value*]

initial-value :
special-value
| *string*
| [**-**] *number*
| **(** *constant-expression* **)**
| *built-in-function* **(** *constant-expression* **)**
| **NULL**

special-value :
CURRENT {
DATABASE
| **DATE**
| **PUBLISHER**
| **TIME**
| **TIMESTAMP**
| **USER**
| **UTC TIMESTAMP** }
| **USER**

パラメーター

- **OR REPLACE 句**　OR REPLACE 句を指定すると、指定された変数がすでに存在する場合は、その変数が削除され、定義が置き換えられます。OR REPLACE 句は、SQL スクリプトの VAREXISTS 関数の代わりに使用できます。

- **DEFAULT 句**　*initial-value* を指定する場合は、*data-type* で定義したデータ型とデータ型を一致させてください。

備考

CREATE VARIABLE 文は、指定したデータ型の新しい変数を作成します。*initial-value* を指定すると、変数はその値に設定されます。*initial-value* を指定しなかった場合、SET 文によって別の値が割り当てられるまで、変数には NULL 値が入っています。

変数は、カラム名を使用できる場所なら SQL 式のどこででも使うことができます。名前の決定は次のように実行されます。

1. クエリの SELECT リストで指定した任意のエイリアスと一致。
2. 任意の参照先テーブルのカラム名と一致。
3. 名前が変数であると仮定。

変数は現在の接続に属し、データベースから切断するか、または DROP VARIABLE 文を使用するまで保持されます。変数は、他の接続からは参照できません。変数は COMMIT または ROLLBACK 文の影響を受けません。

変数は、Embedded SQL プログラムから INSERT または UPDATE 文の大きなテキストまたはバイナリオブジェクトを作成するときに役立ちます。

プロシージャーとトリガーのローカル変数は、複合文の中で宣言します。

パーミッション

なし

関連する動作

なし

参照

- 「複合文」『SQL Anywhere サーバー SQL の使用法』
- 「BEGIN 文」497 ページ
- 「SQL データ型」85 ページ
- 「DROP VARIABLE 文」736 ページ
- 「SET 文」927 ページ
- 「VAREXISTS 関数 [その他]」387 ページ
- 「特別値」61 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

この例は、VARCHAR(50) データ型の変数 first_name を作成します。

```
CREATE VARIABLE first_name VARCHAR(50);
```

この例は、DATE データ型の変数 birthday を作成します。

```
CREATE VARIABLE birthday DATE;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

この例は、初期設定が 5 の INT データ型の変数 v1 を作成します。

```
CREATE VARIABLE v1 INT = 5;
```

この例は、変数 v1 がすでに存在しているかどうかに関係なく、変数 v1 を作成し、値を 10 に設定します。

```
CREATE OR REPLACE VARIABLE v1 INT = 10;
```

CREATE VIEW 文

データベース上にビューを作成します。

構文

CREATE [**OR REPLACE**] **VIEW**
[*owner.*]*view-name* [**(** *column-name*, ... **)**]
AS *query-expression*
[**WITH CHECK OPTION**]

パラメーター

OR REPLACE 句　OR REPLACE (CREATE OR REPLACE VIEW) を指定すると、新しいビューが作成されるか、同じ名前の既存のビューが置き換えられます。OR REPLACE 句を使用する場合、既存のパーミッションは保持されますが、ビューの INSTEAD OF トリガーは削除されます。

1 つ以上の INSTEAD OF トリガーがあるビューで CREATE OR REPLACE VIEW 文を実行すると、エラーが返されます。トリガーを削除してからでないと、ビューを削除または変更できません。

AS 句　ビューのベースとなる SELECT 文です。SELECT 文はローカルテンポラリテーブルを参照してはなりません。また、*query-expression* には GROUP BY 句、HAVING 句、WINDOW 句、または ORDER BY 句を指定でき、UNION、EXCEPT、INTERSECT または共通テーブル式を含めることができます。

クエリのセマンティックは、クエリが ORDER BY 句を SELECT 文の TOP または FIRST 句と組み合わせないと、返されるローの順序が定義されないことを記述します。ORDER BY 句が TOP または FIRST 句のいずれかと共に指定されている場合、クエリはそれぞれ上位 *n* ロー、または 1 番目のローを返します。

WITH CHECK OPTION 句　WITH CHECK OPTION 句を指定すると、*query-expression* で定義された条件を満たさないビューへの更新や挿入が拒否されます。

備考

ビューは物理的にはテーブルとしてデータベースの中に存在しません。ビューは使用するたびに引き出されます。ビューは、CREATE VIEW 文で指定された SELECT 文の結果として引き出されます。ビューの中で使用するテーブル名は、テーブルを同じ名前のテーブルと区別するために、テーブルの所有者のユーザー ID を指定する必要があります。別のユーザーが所有するビューを作成するには、DBA 権限を持ち、所有者を指定する必要があります。

SELECT、DELETE、UPDATE、INSERT 文の中では、ビュー名をテーブル名の代わりに使用できます。

SELECT * は、CREATE VIEW 文のメインクエリでのみ使用できます。抽出テーブルとサブクエリは、* 演算子ではなく、SELECT リストの完全な式を使用する必要があります。たとえば、CREATE VIEW V AS SELECT * FROM T, (SELECT * FROM R) AS DT は、抽出テーブル DT が指定された式による SELECT リストではなく SELECT * を使用しているため、正しくありません。同様に、抽出テーブルで暗黙の SELECT * を使用することはできません。たとえば、CREATE VIEW V AS SELECT * FROM T, LATERAL(proc(T.A.)) AS DT には、LATERAL(proc(T.A.)) が LATERAL(SELECT * FROM proc(T.A.)) の省略形なので暗黙の SELECT * があるため、ビューの定義で使用できません。

ビューを定義する *query-expression* が GROUP BY 句、WINDOW 句、集合関数を含まないか、または集合演算子 (UNION、INTERSECT、EXCEPT) を伴わない場合は、ビューを更新できます。ビューを更新すると、基本となるテーブルも更新されます。

ビューのカラムには *column-name* リスト内で指定した名前が与えられます。column-name リストを指定しないと、ビューのカラムは SELECT リスト項目から名前が与えられます。SELECT リストのすべての項目がユニークな名前になるように指定します。SELECT リスト項目からの名前を使用するには、各項目を単純なカラム名にするか、エイリアス名を指定します。

SQL Anywhere では、*query-expression* の SELECT リスト内の名前のない式を、CREATE VIEW 文で参照できます。*query-expression* の SELECT リストの名前のない式には、名前 **expression** が割り当てられ、このような式が複数存在する場合、整数値と連結されます。たとえば、次の文は 3 つのカラム (expression、expression1、および expression2) を持つビュー V を定義し、これらの名前は作成されたビュー V の SYSCOLUMN システムビューに表示されます。

```
CREATE VIEW V AS
  SELECT DATEADD( DAY, 1, NOW() ), DATEADD( DAY, 2, NOW() ), DATEADD( DAY, 2, NOW() )
  FROM SYS.DUMMY;
```

名前のない SELECT リスト式を持つ他のビューにも同一の名前が割り当てられるため、これらの生成された名前に頼ることはおすすめしません。

一般的に、ビューはカタログに定義されているテーブルやビュー (およびその属性) を参照します。一方、ビューは SQL 変数も参照できます。この場合、ビューを参照するクエリが実行されると、SQL 変数の値が使用されます。SQL 変数を参照するビューは、変数がビューを実行するためのパラメーターとして機能することから、「パラメーター化されたビュー」と呼ばれます。

パラメーター化されたビューは、等価の SELECT ブロックの本体を FROM 句内の派生テーブルとしてクエリに埋め込む手段を提供します。パラメーター化されたビューは、ビューで参照されている SQL 変数を入力パラメーターとして取るようなストアドプロシージャーに埋め込まれたクエリで便利です。

CREATE VIEW 文が実行されるときに SQL 変数が存在している必要はありません。ただし、ビューを参照するクエリを実行するときに SQL 変数が定義されていないと、カラム (変数) が見つからないことを示すエラーが返されます。

パーミッション

ビュー定義内のテーブルに対する RESOURCE 権限と SELECT パーミッションが必要です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

オートコミット。

参照

- 「CREATE TABLE 文」651 ページ
- 「CREATE MATERIALIZED VIEW 文」576 ページ
- 「ALTER MATERIALIZED VIEW 文」427 ページ
- 「INSTEAD OF トリガー」『SQL Anywhere サーバー SQL の使用法』
- 「SELECT 文」895 ページ
- 「ビュー」『SQL Anywhere サーバー SQL の使用法』
- 「通常のビューの作成 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』
- 「ALTER VIEW 文」484 ページ

標準と互換性

- **SQL/2008**　CREATE VIEW は SQL/2008 標準のコア機能ですが、SELECT 文に組み込まれている一部のビュー機能はオプションの言語機能です。ビュー定義で最上位レベルの SELECT 文で ORDER BY 句を指定する機能は、オプションの SQL/2008 言語機能 F852 です。ビューの結果セットを SELECT TOP または LIMIT を使用して制限する機能は、オプションの SQL/2008 言語機能 F859 です (SQL/2008 標準では、この操作を行うために FETCH 句を使用します)。ビューの SELECT 文に、集合、DISTINCT が関係する派生テーブル、または集合演算子 (INTERSECT、EXCEPT、UNION) が含まれているなど、単純に更新できないビュー における WITH CHECK OPTION の指定は、オプションの SQL/2008 言語機能 T111 です。

  CREATE VIEW の一部の機能は、ベンダー拡張です。パラメーター化されたビューは、オプションの OR REPLACE 構文や名前のない SELECT リスト式の名前の自動生成と同様に、ベンダー拡張です。

例

次の例は、男性従業員の情報のみを表示するビューを作成します。このビューはベーステーブルと同じカラム名を持ちます。

```
CREATE VIEW MaleEmployees
AS SELECT *
FROM Employees
WHERE Sex = 'M';
```

次の例は、従業員と所属する部署を表示するビューを作成します。

```
CREATE VIEW EmployeesAndDepartments
  AS SELECT Surname, GivenName, DepartmentName
  FROM Employees JOIN Departments
  ON Employees.DepartmentID = Departments.DepartmentID;
```

次の例は、前の例で作成された EmployeesAndDepartments ビューを置き換えます。置き換え後のビューには、各従業員の住所の市、州、国が表示されます。

```
CREATE OR REPLACE VIEW EmployeesAndDepartments
  AS SELECT Surname, GivenName, City, State, Country
  FROM Employees JOIN Departments
  ON Employees.DepartmentID = Departments.DepartmentID;
```

次の例は、パラメーター化されたビューを変数 var1 と var2 に基づいて作成します。この 2 つはどちらも Employees テーブルまたは Departments テーブルの属性ではありません。

```
CREATE VIEW EmployeesByState
  AS SELECT Surname, GivenName, DepartmentName
  FROM Employees JOIN Departments
  ON Employees.DepartmentID = Departments.DepartmentID
  WHERE Employees.State = var1 and Employees.Status = var2;
```

変数は、変数が式として許可されている任意のコンテキストで、ビューの SELECT 文に出現させることができます。たとえば、次のパラメーター化されたビューでは、パラメーター var1 を LIKE 述部のパターンとして使用しています。

```
CREATE VIEW ProductsByDescription
  AS SELECT *
  FROM Products
  WHERE Products.Description LIKE var1;
```

このビューを使用するには、ビューを参照するクエリの実行前に変数 var1 が定義されている必要があります。たとえば、次の BEGIN 文はプロシージャー、関数、またはバッチ文に使用できます。

```
BEGIN
DECLARE var1 CHAR(20);
SET var1 = '%cap%';
SELECT * FROM ProductsByDescription
END
```

DEALLOCATE DESCRIPTOR 文 [ESQL]

SQL 記述子領域に関連付けられているメモリを解放します。

構文

DEALLOCATE DESCRIPTOR *descriptor-name*

descriptor-name : *identifier*

備考

データ項目、インジケーター変数、構造体そのものなど、記述子領域に関連付けられているすべてのメモリを解放します。

パーミッション

なし

関連する動作

なし

参照

- 「ALLOCATE DESCRIPTOR 文 [ESQL]」406 ページ
- 「SQLDA (SQL descriptor area)」『SQL Anywhere サーバー プログラミング』
- 「SET DESCRIPTOR 文 [ESQL]」907 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　DEALLOCATE DESCRIPTOR は、SQL/2008 のオプションの言語機能 B031、"Basic dynamic SQL" の一部です。

例

例については、「ALLOCATE DESCRIPTOR 文 [ESQL]」406 ページを参照してください。

DEALLOCATE 文

この文は SQL Anywhere では機能しないので、無視されます。これは Adaptive Server Enterprise と Microsoft SQL Server との互換性のために用意されています。この文の詳細については、Adaptive Server Enterprise または Microsoft SQL Server のマニュアルを参照してください。

標準と互換性

- **SQL/2008**　ベンダー拡張。

宣言セクション [ESQL]

Embedded SQL プログラムのホスト変数を宣言します。ホスト変数を使って、データベースとデータを交換します。

構文

EXEC SQL BEGIN DECLARE SECTION;
C declarations
EXEC SQL END DECLARE SECTION;

備考

宣言セクションは、BEGIN DECLARE SECTION と END DECLARE SECTION 文で囲まれた C 変数宣言のセクションです。宣言セクションを使うと、SQL プリプロセッサーはホスト変数として使われる C 変数を認識します。すべての C 宣言が宣言セクションの中で有効なわけではありません。

パーミッション

なし

参照

- 「Embedded SQL のホスト変数」『SQL Anywhere サーバー プログラミング』
- 「BEGIN 文」497 ページ

標準と互換性

- **SQL/2008**　コア機能。

例

```
EXEC SQL BEGIN DECLARE SECTION;
char *surname, initials[5];
int dept;
EXEC SQL END DECLARE SECTION;
```

DECLARE CURSOR 文 [ESQL] [SP]

カーソルを宣言します。

構文 1 [ESQL]

DECLARE *cursor-name*
[**UNIQUE**]
[**NO SCROLL**
| **DYNAMIC SCROLL**
| **SCROLL**
| **INSENSITIVE**
| **SENSITIVE**
]
CURSOR FOR
{ *select-statement*
| *statement-name*
| *call-statement* }

構文 2 [SP]

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*

パラメーター

UNIQUE 句　カーソルが UNIQUE として宣言されている場合、クエリは各ローをユニークに識別するのに必要なすべてのカラムを強制的に返します。このため、プライマリキーまたはユニークなテーブル制約内のカラムのすべてが、確実に返ることになります。必須であってもクエリに指定されていなかったカラムは、結果セットに追加されます。

UNIQUE カーソル上で実行された DESCRIBE は、インジケーター変数の中に次の追加のオプションを設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **DT_KEY_COLUMN**　カラムはローに対するキーの一部です。

○ **DT_HIDDEN_COLUMN**　カラムがクエリに追加されました。カラムはローをユニークに識別するために必要です。

NO SCROLL 句　NOSCROLL として宣言されたカーソルは、FETCHNEXT と FETCHRELATIVE0 の各シーク操作を使用して、結果セットの前方移動に制限されます。

カーソルがローを出た後は、そのローに戻ることができないため、カーソルに sensitivity の制限はありません。NO SCROLL カーソルが要求されると、SQL Anywhere は最も効率的なカーソルである asensitive カーソルを提供します。

DYNAMIC SCROLL 句　DYNAMIC SCROLL はデフォルトカーソルタイプです。DYNAMIC SCROLL カーソルでは、FETCH 文のすべてのフォーマットを使用できます。

DYNAMIC SCROLL カーソルを要求されると、SQL Anywhere は asensitive カーソルを提供します。カーソルを使用する場合、効率と一貫性の間には常にトレードオフ関係があります。Asensitive カーソルを使用すると、パフォーマンスは向上しますが一貫性が低下します。

SCROLL 句　SCROLL として宣言されたカーソルは、FETCH 文のすべてのフォーマットを使用できます。SCROLL カーソルを要求されると、SQL Anywhere は value-sensitive カーソルを提供します。value-sensitive カーソルを使用すると、基本となるローが変更または削除されている場合に、以前に FETCH された結果ローの後に続く FETCH に対して、警告またはエラーが返されることがあります。

SQL Anywhere は、結果セットのメンバーシップが保証されるような方法で value-sensitive カーソルを実行します。DYNAMIC SCROLL カーソルの方が効率的です。SCROLL カーソルの一貫した動作が必要でない場合は、DYNAMIC SCROLL カーソルを使用してください。

INSENSITIVE 句　INSENSITIVE として宣言されたカーソルには、それを開いたときに決定されるメンバーシップがあります。テンポラリテーブルは、すべてのオリジナルローのコピーから作成されます。INSENSITIVE カーソルからの FETCH では、同時に実行されているトランザクションの別の INSERT、UPDATE、または DELETE 文、および同じトランザクション内からの別の更新操作の影響を受けません。INSENSITIVE カーソルは更新できません。

SENSITIVE 句　SENSITIVE として宣言されたカーソルは、結果セットのメンバーシップまたは値の変更に依存します。

***statement-name* 句の場合**　PREPARE 文を使用して文に名前を付けます。作成された SELECT または CALL のためにだけカーソルを宣言できます。SQL プリプロセッサーの -m HISTORICAL オプションが指定されている場合を除き、PREPARE 文に指定されているカーソル更新可能性がカーソルに使用されます。

***variable-name* 句の使用**　ストアドプロシージャーの内部でのみ使用します。この変数はカーソルの SELECT 文を含む文字列です。この変数は DECLARE を処理するときに使用するため、次のいずれかの方法で指定します。

プロシージャーへのパラメーター。次に例を示します。

```
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;
```

この変数に値を割り当ててから、別の BEGIN...END 内にネストする。次に例を示します。

```
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;
```

備考

カーソルはクエリの結果を操作する主要な手段です。DECLARE CURSOR 文は、SELECT 文または CALL 文に対して指定された名前を持つカーソルを宣言します。Watcom SQL のプロシージャー、トリガー、またはバッチの場合、DECLARE CURSOR 文は、BEGIN キーワードの直後に、他の宣言とともに指定してください。カーソル名はユニークにしてください。

カーソルが複合文内で宣言される場合、(Watcom SQL または Transact-SQL 複合文のいずれで宣言されていても) その複合文を実行している間のみ存在します。

DECLARE CURSOR 文で指定したカーソルのタイプで、その文のクエリオプティマイザーによって選択される実行プランを決定できます。たとえば、SELECT 文の INSENSITIVE カーソルは、カーソルを開いたときに SELECT 文の結果セットが完全に実体化されることを要求します。また、カーソルのタイプは基本となる文の特性と一致している必要があります。カーソルタイプと文に不一致があった場合、カーソルタイプが自動的に変更されることがあります。たとえば、定義上 INSENSITIVE カーソルは読み込み専用のため、INSENSITIVE カーソル宣言は FOR UPDATE を指定している更新可能な SELECT 文とは競合します。このような場合、カーソルタイプは、カーソルが開かれたときに自動的に INSENSITIVE から更新可能な value-sensitive カーソルに変更されます。

カーソル宣言に組み込まれている SELECT 文の更新可能性は、指定されていない場合 ansi_update_constraints オプションの設定によって決定されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

なし

関連する動作

基本となる文の要件を満たすようにカーソルタイプを変更する必要がある場合、カーソルが開かれたときに警告が返されます。

参照

- 「PREPARE 文 [ESQL]」856 ページ
- 「OPEN 文 [ESQL] [SP]」844 ページ
- 「EXPLAIN 文 [ESQL]」747 ページ
- 「SELECT 文」895 ページ
- 「CALL 文」503 ページ
- 「FOR 文」753 ページ
- FOR UPDATE または FOR READ ONLY 句、SELECT 文900 ページ
- WITH table-hint 句、FROM 句764 ページ
- 「SQL プリプロセッサー」『SQL Anywhere サーバー プログラミング』
- 「sensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「insensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「value-sensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「asensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「ansi_update_constraints オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　DECLARE CURSOR は、SQL/2008 標準のコア機能です。FOR UPDATE に SCROLL または NO SCROLL を付けて指定する機能は、オプションの SQL 言語機能 F831、"Full cursor update" です。Embedded SQL プログラムでの DECLARE CURSOR の使用は、オプションの SQL 言語機能 B031 によって構成されています。いくつかのカーソルタイプもオプションの SQL 機能です。これらのコードを以下に示します。

  ○INSENSITIVE カーソルは、SQL/2008 標準のオプションの SQL 言語機能 F791 です。

  ○SENSITIVE カーソルは、SQL/2008 標準のオプションの SQL 言語機能 F231 です。

  ○スクロール可能カーソルは、SQL/2008 標準のオプションの SQL 言語機能 F431 です。

  SQL Anywhere では、以下のような DECLARE CURSOR の多くのベンダー拡張がサポートされます。

  ○SQL Anywhere では、SQL/2008 において DECLARE CURSOR 文の句として定義されている FOR UPDATE 句の複数の拡張がサポートされます。

  ○WITH HOLD は、SQL/2008 の定義に従った DECLARE CURSOR 文の句としてではなく、OPEN 文の句として指定されます。

  ○SQL/2008 標準では、カーソルの感知性とスクロール動作の概念を区別していますが、SQL Anywhere では、歴史的な理由から、この 2 つを組み合わせています。SQL Anywhere では、NO SCROLL として宣言されているカーソルを除き、すべてのカーソルは前後にスクロール可能です。

○DYNAMIC SCROLL と UNIQUE はベンダー拡張です。DYNAMIC SCROLL は、SQL/2008 標準の ASENSITIVE として宣言されるカーソルと似た動作をします。

○CALL 文を介して、または USING 句を使用してカーソルを宣言する機能は、ベンダー拡張です。

● **Transact-SQL** DECLARE CURSOR は、Adaptive Server Enterprise によってサポートされますが、いくつかの動作が異なります。Adaptive Server Enterprise は、SQL/2008 と同様に、スクロール動作と感知性を区別します。Adaptive Server Enterprise では、カーソル感知性オプションは SEMI-SENSITIVE、INSENSITIVE、またはデフォルト (ASENSITIVE と類似) です。Adaptive Server Enterprise では、NO SCROLL カーソルがデフォルトであり、スクロール可能なカーソルはすべて読み込み専用です。DECLARE CURSOR 文のいくつかの機能は、Adaptive Server Enterprise ではサポートされません。これらのコードを以下に示します。

○Adaptive Server Enterprise では、SQL Anywhere のカーソル同時実行性句はサポートされません。

フェッチされるローに対するロックを取得するには、HOLDLOCK テーブルヒントを使用してください。

○Adaptive Server Enterprise は、DYNAMIC SCROLL カーソルおよび UNIQUE カーソルをサポートしません。DYNAMIC SCROLL は、Adaptive Server Enterprise のデフォルトカーソルの動作と似ています。

○CALL 文を介して、または USING 句を使用してカーソルを宣言する機能は、Adaptive Server Enterprise ではサポートされません。

Adaptive Server Enterprise では、Transact-SQL プロシージャーとファンクションに対して、同じカーソル名を使用する複数の DECLARE CURSOR 文を指定できます。Adaptive Server Enterprise では、DEALLOCATE CURSOR 文は、後続の OPEN 文が以前宣言された正しいカーソルを参照できるように、現在のスコープからカーソルを削除するために使用されます。この機能は SQL Anywhere ではサポートされていません。SQL Anywhere では、指定されたスコープ内のすべてのカーソルは、ユニークな名前を持つ必要があります。Transact-SQL ダイアレクトのプロシージャーに同じ名前の複数のカーソル宣言が含まれている場合、プロシージャーが解析されるとき、エラーは発生しません。ただし、実行時に、同じカーソル名の 2 つ目の DECLARE CURSOR 文が実行されると、エラーが発生します。

Open Client 接続および jConnect 接続の TDS Wire Protocol は、完全にスクロール可能な結果セットを実装しないことに注意してください。カーソルで後方にスクロールするときに、TDS クライアントによってすでに取り出されている、プリフェッチされたローのウィンドウ内に必要なローがある場合、FETCH 要求はすぐに履行される可能性があります。ただし、必要なローがこのウィンドウの外にある場合、カーソルの SELECT 文が再実行されることがあります。

例

次の例は、Embedded SQL 内のスクロールカーソルを宣言する方法を示します。

```
EXEC SQL DECLARE cur_employee SCROLL CURSOR
FOR SELECT * FROM Employees;
```

次の例は、Embedded SQL 内の準備文のためのカーソルを宣言する方法を示します。

```
EXEC SQL PREPARE employee_statement
FROM 'SELECT Surname FROM Employees'FOR READ ONLY;
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement;
```

次の例は、ストアドプロシージャーのカーソルの使用方法を示します。

```
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
```

DECLARE LOCAL TEMPORARY TABLE 文

ローカルテンポラリテーブルを宣言します。

構文

DECLARE LOCAL TEMPORARY TABLE *table-name*
({ *column-definition* [*column-constraint* …] | *table-constraint* | *pctfree* }, … **)**
[**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS**
  | **NOT TRANSACTIONAL**]

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

パラメーター

ON COMMIT 句　デフォルトでは、テンポラリテーブルのローは COMMIT のときに削除されます。ON COMMIT 句を使用すると、COMMIT のときにローを保護できます。

NOT TRANSACTIONAL 句　この句を使用して作成されたテーブルは、COMMIT と ROLLBACK のいずれの影響も受けません。状況によっては、NOT TRANSACTIONAL 句を使用するとパフォーマンスが向上します。これは、トランザクション単位でないテンポラリテーブルでの操作では、ロールバックログにエントリが作成されないためです。たとえば、テンポラリテーブルを使用するプロシージャーが COMMIT や ROLLBACK の介入を受けずに繰り返し呼び出される場合、NOT TRANSACTIONAL が有用です。

備考

REFERENCES *column-constraint* および FOREIGN KEY table-constraint は、ローカルテンポラリテーブルに対して使用できません。

DECLARE LOCAL TEMPORARY TABLE 文はテンポラリテーブルを宣言します。

DECLARE LOCAL TEMPORARY TABLE を使用して作成されたテーブルは、システムカタログの SYSTABLE ビューに表示されません。

宣言されたテンポラリテーブルのローは、テーブルが明示的に削除されたとき、またはスコープ外になったときに削除されます。TRUNCATE または DELETE を使用して、明示的にローを削除することもできます。

複合文内で宣言されたローカルテンポラリテーブルは、複合文内に存在します。それ以外の場合、宣言されたローカルテンポラリテーブルは接続の終わりまで存在します。

同じスコープ内にある 2 つのローカルテンポラリテーブルは、同じ名前にはできません。ベーステーブルと同じ名前のテンポラリテーブルを作成すると、ローカルテンポラリテーブルのスコープが終了した時点で、そのベーステーブルは、その接続内でのみ参照できるようになります。接続では、既存のテンポラリテーブルと同じ名前のベーステーブルを作成できません。

プロシージャーの完了後も保持されるローカルテンポラリテーブルをプロシージャーで作成するには、代わりに CREATE LOCAL TEMPORARY TABLE 文を使用します

パーミッション

なし

関連する動作

なし

参照

- 「CREATE TABLE 文」651 ページ
- 「CREATE LOCAL TEMPORARY TABLE 文」571 ページ
- 「複合文」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　DECLARE LOCAL TEMPORARY TABLE は、SQL/2008 標準のオプションの言語機能 F531 の一部です。PCTFREE 句と NOT TRANSACTIONAL 句はベンダー拡張です。また、文によって定義されたカラムと制約の定義にも、ベンダー拡張の構文を含めることができます。SQL/2008 では、DECLARE LOCAL TEMPORARY TABLE 文によって作成されるテーブルをシステムカタログに表示することが規定されていますが、SQL Anywhere では表示されません。

- **Transact-SQL**　DECLARE LOCAL TEMPORARY TABLE は、Adaptive Server Enterprise ではサポートされません。Sybase Adaptive Server Enterprise では、特殊文字 # で始まるテーブル名を使用して CREATE TABLE 文でテンポラリテーブルを作成します。

例

次の例は、ストアドプロシージャー内のテンポラリテーブルの宣言方法を示します。

```
BEGIN
 DECLARE LOCAL TEMPORARY TABLE TempTab ( number INT );
 ...
END
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DECLARE 文

複合文 (BEGIN...END) 内で SQL 変数または式を宣言します。

構文 1：変数の宣言

DECLARE *variable-name* [, ...] *data-type*
[{ **=** | **DEFAULT** } *initial-value*]

initial-value :
special-value
| *string*
| [**-**] *number*
| **(** *constant-expression* **)**
| *built-in-function* **(** *constant-expression* **)**
| **NULL**

special-value :
CURRENT {
DATABASE
| **DATE**
| **PUBLISHER**
| **TIME**
| **TIMESTAMP**
| **USER**
| **UTC TIMESTAMP** }
| **USER**

構文 2：例外の宣言

DECLARE *exception-name* **EXCEPTION**
FOR SQLSTATE [**VALUE**] *string*

備考

「DECLARE *variable-name* : 」プロシージャー、トリガー、またはバッチの本体で使用される変数は DECLARE 文で構文 1 を使用して宣言できます。変数は、宣言される複合文の間持続します。*initial-value* を指定すると、変数はその値に設定されます。*initial-value* を指定しなかった場合、SET 文によって別の値が割り当てられるまで、変数には NULL 値が入っています。

Watcom SQL プロシージャーの本文またはトリガーは複合文です。また、カーソルの宣言 (DECLARE CURSOR) など他の宣言では、BEGIN キーワードの直後に変数を宣言する必要があります。Transact-SQL プロシージャーまたはトリガーには、そのような制約はありません。

initial-value を指定する場合は、*data-type* で定義したデータ型とデータ型を一致させてください。

「DECLARE *exception-name* EXCEPTION : 」この構文を使用して、複合文 (BEGIN...END) 内で SQL 言語の例外の変数を宣言します。この変数は、たとえば、実行中に取得される SQLSTATE との比較に使用したり、SIGNAL 文で使用したり、例外ハンドラー内の例外ケースの一部として使用したりできます。

参照

- 「SQL データ型」85 ページ
- 「DECLARE 文」693 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「BEGIN 文」497 ページ
- 「SQLSTATE 特別値」71 ページ
- 「特別値」61 ページ
- 「例外ハンドラー」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** 「構文 1」 (変数の宣言) - 永続的ストアドモジュール機能。「構文 2」 (例外の宣言) - SQL Anywhere でサポートされている例外宣言の形式 (つまり DECLARE EXCEPTION 文) はベンダー拡張です。SQL/2008 では、例外はキーワード DECLARE HANDLER を使用するハンドラー宣言で指定されます。DECLARE...EXCEPTION 構文は、T-SQL プロシージャーでは使用できません。

- **Transact-SQL** 「構文 2」 (例外の宣言) は、Transact-SQL 複合文およびプロシージャーでは使用できません。

例

次のバッチは、DECLARE 文の使用法を示し、メッセージをデータベースサーバーメッセージウィンドウに表示します。

```
BEGIN
  DECLARE varname CHAR(61);
  SET varname = 'Test name';
  MESSAGE varname;
END
```

次の例は、以下の変数を宣言します。

- INT、初期設定値 5 の v1。
- CHAR(10)、初期値 abc の v2 と v3。

```
BEGIN
   DECLARE v1 INT = 5;
   DECLARE v2, v3 CHAR(10) = 'abc';
          // ...
END
```

次のプロシージャーは SQLSTATE 比較で使用する例外を宣言します。

```
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;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    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;
```

次の複合文は、SIGNAL および例外ハンドラーで使用する例外を宣言します。

```
  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;
```

DELETE 文 (位置付け) [ESQL] [SP]

カーソルの現在位置のデータを削除します。

構文

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*

備考

この形式の DELETE 文は、指定されたカーソルの現在のローを削除します。現在のローは、カーソルからフェッチされた最後のローと定義されます。

ローを削除するテーブルは次のように決定します。

- FROM 句が含まれない場合、カーソルは単一テーブルだけにあります。
- カーソルがジョインされたクエリ用の場合 (ジョインがあるビューの使用を含めて)、FROM 句が使われます。指定したテーブルの現在のローだけが削除されます。ジョインに含まれた他のテーブルは影響を受けません。
- FROM 句が含まれている場合は、*table* によって、カーソル内の更新可能なテーブルが明確に指定されている必要があります。*correlation-name* が指定されている場合、サーバーは基本となるカーソルで指定されている相関名とその相関名を照合しようとします。DELETE 文に相関名が指定されておらず、テーブル所有者が指定されていない場合、サーバーは *table-or-view* と基本となるカーソルの更新可能なテーブルを照合しようとします。*table-or-view* は最初に相関名に対して照合されます。
  - 相関名が基本となるカーソルに存在する場合、*table-or-view* は対応する相関名と照合されることがあります。
  - 相関名が存在しない場合、*table-or-view* はカーソルのテーブル名と明確に一致する必要があります。
- FROM 句が含まれており、かつテーブル所有者が指定されている場合、*table* は基本となるカーソルの更新可能なテーブルと明確に一致する必要があります。
- 位置付け DELETE 文はビューでカーソルを開くときに使用できます。ただし、ビューが更新可能である場合にかぎられます。

パーミッション

カーソル内で使用されるテーブルに対する DELETE パーミッションが必要です。

関連する動作

なし

参照

- 「UPDATE 文」976 ページ
- 「UPDATE (位置付け) 文 [ESQL] [SP]」971 ページ
- 「INSERT 文」802 ページ
- 「PUT 文 [ESQL]」860 ページ

標準と互換性

- **SQL/2008**　DELETE 文 (位置付け) は、SQL/2008 標準のコア機能です。Embedded SQL プログラム内から位置付け DELETE 文を使用する機能は、オプションの SQL 言語機能 B031、"Basic dynamic SQL" の一部です。

  SQL/2008 では、FROM キーワードは必須ですが、SQL Anywhere ではオプションです。ansi_update_constraints オプションが Off に設定されている場合、更新可能なカーソルの範囲にはベンダー拡張が含まれます。

例

次の文は、データベースから カーソル cur_employee の現在のローを削除します。

```
DELETE
WHERE CURRENT OF cur_employee;
```

DELETE 文

データベースからローを削除します。

構文 1

DELETE [*row-limitation*]
[**FROM**] [*owner*.]*table-or-view* [[**AS**] *correlation-name*]
[**WHERE** *search-condition*]
[**ORDER BY** { *expression* | *integer* } [**ASC** | **DESC**], ...]
[**OPTION(** *query-hint*, ... **)**]

構文 2 - Transact-SQL

DELETE [*row-limitation*]
[**FROM**] [*owner*.]*table-or-view* [[**AS**] *correlation-name*]
[**FROM** *table-expression*]
[**WHERE** *search-condition*]
[**ORDER BY** { *expression* | *integer* } [**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 : ジョインを含む完全なテーブル式。「FROM 句」758 ページを参照してください。

option-name :*identifier*

option-value :
hostvar (許可されたインジケーター)
| *string*
| *identifier*
| *number*

パラメーター

***row-limitation* 句**　ローを制限する句を使用することによって、削除されるローを WHERE 句を満たすローのサブセットのみに制限できます。TOP 引数と START AT 引数には、ホスト変数、整数定数、または整数変数を使用した簡単な算術演算を指定できます。TOP 引数は 0 以上である必要があります。START AT 引数は 0 より大きい値にする必要があります。これらの句を指定する場合は、ローの順序を意味のあるものにするために ORDER BY 句も指定します。

FROM 句　FROM 句は、ローが削除されるテーブルを示します。構文 2 では、DELETE 文の第 2 FROM 句によって、他のテーブルとのジョインに基づいて指定されたテーブルから削除されるローが決定されます。*table-expression* には、任意の複雑なテーブル式 (抽出テーブル、KEY ジョイン、および NATURAL ジョインを含む) を指定できます。

次の例は、構文 2 が使用されたときに、相関名が照合される方法を示します。次の文では、

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_2 AS alias_2
WHERE ...
```

テーブル table_1 の相関名は、第 1 FROM 句にはありませんが、第 2 FROM 句には含まれています。この場合、第 1 FROM 句の中の table_1 は、第 2 FROM 句では alias_1 で識別されます。この文の中に table_1 のインスタンスは 1 つしかありません。これは、同じ文の中で相関名を使用する方法と使用しない方法の両方を使ってテーブルを識別する場合、テーブルの 2 つのインスタンスが考慮されるという一般規則の例外として許可されています。

ただし、次の例では、第 2 FROM 句に table_1 のインスタンスが 2 つあります。この文は構文エラーで失敗します。第 2 FROM 句の table_1 のインスタンスのうち、どれが第 1 FROM 句の table_1 の最初のインスタンスと一致するのかが明確ではないためです。

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

WHERE 句　DELETE 文は、WHERE 句の条件を満たすすべてのローを削除します。WHERE 句を指定しない場合、指定したテーブルからすべてのローは削除されます。第 2 FROM 句がある場合、WHERE 句はこの第 2 FROM 句の *table-expression* のローの条件を与えます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ORDER BY 句　削除されるローのソート順を指定します。通常、ローを更新する順序は重要ではありません。ただし、FIRST 句または TOP 句と一緒に使用する場合は、ローを更新する順序が意味を持ちます。

ORDER BY 句ではカラムの順序数を使用できません。

ORDER BY リストの各項目には、昇順の場合 (デフォルト) は ASC、降順の場合は DESC のラベルを付けることができます。

OPTION 句　この句は、文を実行するためのヒントを指定するときに使用します。次のヒントがサポートされます。

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- *option-name* = *option-value* クエリテキスト内の OPTION(isolation_level = ...) の指定は、クエリの独立性レベルを指定する他のいずれの手段よりも優先されることに注意してください。

備考

DELETE 文を使用してデータを大量に削除する場合も、カラム統計は更新されます。

テーブルのすべてのローを削除する場合、より効率的な TRUNCATE TABLE 文の使用を検討してください。

ビューを定義しているクエリ指定が更新可能な場合は、ビューに対して DELETE 操作を実行できます。ビューは、ビューを定義している SELECT 文の FROM 句のテーブルが 1 つのみであり、かつ、DISTINCT 句、GROUP BY 句、WINDOW 句、集合関数、および集合演算子 (UNION または INTERSECT など) を含んでいない場合に、更新可能です。

パーミッション

テーブルに対する DELETE パーミッションが必要です。

関連する動作

なし

参照

- 「TRUNCATE 文」961 ページ
- 「SELECT、UPDATE、DELETE クエリブロック内のロー制限句」『SQL Anywhere サーバー SQL の使用法』
- 「INSERT 文」802 ページ
- 「INPUT 文 [Interactive SQL]」794 ページ
- 「FROM 句」758 ページ
- OPTION 句、SELECT 文902 ページ
- 「通常のビュー」『SQL Anywhere サーバー SQL の使用法』
- 「削除時のロック」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　構文 1 は SQL/2008 標準のコア機能ですが、構文 2 は Transact-SQL のベンダー拡張です。構文 1 の次の機能はベンダー拡張です。

  ○オプションの FROM キーワード。

  ○*row-limitation* 句と ORDER BY 句。

  ○OPTION 句。

例

FinancialData テーブルから、2000 年より前のデータすべてを削除します。

```
DELETE
FROM FinancialData
WHERE Year < 2000;
```

SalesOrderItems テーブルの最初から 10 の注文を削除します。各注文の出荷日は 2001-01-01 であり、地域は Central です。

```
DELETE TOP 10
FROM SalesOrderItems
FROM SalesOrders
WHERE SalesOrderItems.ID = SalesOrders.ID
  and ShipDate < '2001-01-01' and Region ='Central'
ORDER BY ShipDate ASC;
```

独立性レベル 3 で文を実行し、データベースから部署 600 を削除します。

```
DELETE FROM Departments
WHERE DepartmentID = 600
OPTION( isolation_level = 3 );
```

DESCRIBE 文 [ESQL]

データベースから取り出したデータを格納するために必要なホスト変数、またはデータベースにデータを渡すために必要なホスト変数に関する情報を取得します。

構文

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*

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

cursor-name : *宣言されたカーソル*

sqlda-name : *identifier*

パラメーター

USER TYPES 句　USER TYPES 句を持つ DESCRIBE 文は、カラムのドメインに関する情報を返します。通常、このような DESCRIBE は、以前の DESCRIBE が DT_HAS_USERTYPE_INFO のインジケーターを返したときに行われます。

返された情報は、sqlname フィールドがカラム名の代わりにドメインの名前を保持すること以外は、USER TYPES キーワードのない DESCRIBE と同じです。

DESCRIBE が LONG NAMES 句を使用する場合は、sqldata フィールドがこの情報を保持します。

ALL 句　DESCRIBE ALL を使用すると、データベースサーバーへの 1 回の要求で INPUT と OUTPUT を記述できます。これにはパフォーマンス上の利点があります。OUTPUT 情報が最初に SQLDA に格納され、次に INPUT 情報が格納されます。sqld フィールドには、INPUT と OUTPUT 変数の総数が入ります。インジケーター変数内の DT_DESCRIBE_INPUT ビットは INPUT 変数に対して設定され、OUTPUT 変数に対してクリアします。

BIND VARIABLES FOR 句　INPUT 句と同じです。

SELECT LIST FOR 句　OUTPUT 句と同じです。

INPUT 句　バインド変数は、データベースが文を実行するときにアプリケーションによって提供される値です。バインド変数は、文に対するパラメーターと考えることができます。DESCRIBE INPUT は、バインド変数名を使って SQLDA に名前フィールドを格納します。DESCRIBE INPUT は、SQLDA の sqlda フィールドにバインド変数の数も格納します。

DESCRIBE は、SQLDA の中のインジケーター変数を使って情報を追加します。DT_PROCEDURE_IN と DT_PROCEDURE_OUT は、CALL 文を記述するときに、インジケーター変数の中に設定されるビットです。DT_PROCEDURE_IN は IN または INOUT パラメーターを示し、DT_PROCEDURE_OUT は INOUT または OUT パラメーターを示します。プロシージャー RESULT カラムは、両方のビットをクリアします。DESCRIBE OUTPUT の後、これらのビットは結果セットを持っている文 (OPEN、FETCH、RESUME、CLOSE を使用する必要がある) と持っていない文 (EXECUTE を使用する必要がある) を区別するのに使用できます。DESCRIBE INPUT は、バインド変数が CALL 文に対する引数であるときに、適切に DT_PROCEDURE_IN と DT_PROCEDURE_OUT を設定するだけです。CALL 文の引数である式の中のバインド変数は、ビットを設定しません。

OUTPUT 句　DESCRIBE OUTPUT 文は、SQLDA のそれぞれの SELECT リスト項目のデータ型と長さを埋めます。名前フィールドにも、SELECT リスト項目の名前が入ります。SELECT リスト項目に対してエイリアスを指定する場合、名前はそのエイリアスになります。エイリアスを指定しない場合、名前を SELECT リスト項目から取り出します。項目が単純なカラム名である場合は、その名前が使用されます。そうでない場合は、式の部分文字列が使用されます。また、DESCRIBE は、SELECT リスト項目の数を SQLDA の sqld フィールドに格納します。

記述されている文が 2 つ以上の SELECT 文の UNION である場合、DESCRIBE OUTPUT に対して返されるカラム名は、最初の SELECT 文に対して返されるカラム名と同じです。

CALL 文を記述すると、DESCRIBE OUTPUT 文は、プロシージャー中の各 INOUT または OUT パラメーターに対して、SQLDA の中にデータ型、長さ、名前を格納します。DESCRIBE OUTPUT 文は、SQLDA の sqld フィールドの中に INOUT または OUT パラメーターの数も格納します。

結果セットを返す CALL 文を記述すると、DESCRIBE OUTPUT 文は、プロシージャー定義の各 RESULT カラムに対して、SQLDA の中にデータ型、長さ、名前を格納します。DESCRIBE OUTPUT 文は、SQLDA の sqld フィールドの中に結果カラムの数も格納します。

LONG NAMES 句　LONG NAMES 句は、文またはカーソルに対応するカラム名を取得するために用意されています。この句を使用しないと、取得するカラム名の長さは 29 文字に制限されますが、この句を使用すると、任意の長さのカラム名がサポートされます。

LONG NAMES を使用した場合、カーソルからフェッチを行う場合と同様に、長い名前は SQLDA の SQLDATA フィールドに格納されます。これ以外のフィールド (SQLLEN、SQLTYPE など) にはどれも入力されません。SQLDA は FETCH SQLDA のように設定してください。つまり、各カラムには、文字列型のデータのエントリを 1 つ含めます。インジケーター変数がある場合、通常の方法でトランケーションが示されます。

長い名前のデフォルトの仕様は、**TABLE.COLUMN** です。

WITH VARIABLE RESULT 句　この句は、複数の結果セットと異なる数または型のカラムを持つプロシージャーの記述に使用します。

WITH VARIABLE RESULT を使用する場合、DESCRIBE 文後にデータベースサーバーは SQLCOUNT 値を次のいずれかに設定します。

- **0**　結果セットは変更される場合があります。各 OPEN 文の後でプロシージャーコールを記述し直してください。
- **1**　結果セットは固定です。再度記述する必要はありません。

備考

DESCRIBE 文は指定した SQLDA を設定し、指定した文に対して OUTPUT (SELECT LIST FOR と同等) または INPUT (BIND VARIABLES FOR と同等) を記述します。

INPUT の場合、DESCRIBE BIND VARIABLES は SQLDA の中にデータ型を設定しません。アプリケーションが、データ型を設定する必要があります。ALL キーワードを使うと、1 つの SQLDA の中に INPUT と OUTPUT を記述できます。

文名を指定する場合は、同じ文名で PREPARE 文を使用して、文を事前に作成します。また、事前に SQLDA を割り付けます。

カーソル名を指定する場合、カーソルを事前に宣言し、開いておきます。デフォルトの動作は、OUTPUT です。SELECT 文と CALL 文だけが OUTPUT を持っています。他の文または動的カーソルではないカーソルの DESCRIBE OUTPUT は、SQLDA の sqld フィールドを 0 に設定して出力なしを示します。

Embedded SQL の場合、NCHAR、NVARCHAR、LONG NVARCHAR はそれぞれデフォルトで DT_FIXCHAR、DT_VARCHAR、DT_LONGVARCHAR と記述されます。db_change_nchar_charset

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関数が呼び出された場合、これらのデータ型はそれぞれ DT_NFIXCHAR、DT_NVARCHAR、DT_LONGNVARCHAR と記述されます。

パーミッション

なし

関連する動作

なし

参照

- 「ALLOCATE DESCRIPTOR 文 [ESQL]」406 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「OPEN 文 [ESQL] [SP]」844 ページ
- 「PREPARE 文 [ESQL]」856 ページ
- 「db_change_nchar_charset 関数」『SQL Anywhere サーバー プログラミング』
- 「SQLDA (SQL descriptor area)」『SQL Anywhere サーバー プログラミング』
- 「LONG NVARCHAR データ型」87 ページ
- 「NCHAR データ型」88 ページ
- 「NVARCHAR データ型」89 ページ

標準と互換性

- **SQL/2008**　DESCRIBE OUTPUT 文は、SQL/2008 標準のオプションの SQL 言語機能 B031、"Basic dynamic SQL" です。DESCRIBE INPUT 文は、オプションの SQL 言語機能 B032、"Extended dynamic SQL" です。DESCRIBE 文のその他の多くの句は、ベンダー拡張です。これらのコードを以下に示します。

  ○USER TYPES、ALL、BIND VARIABLES FOR、LONG NAMES、WITH VARIABLE RESULT の各句。

  ○DESCRIBE は INTO 句を使用して SQLDA を識別します。SQL/2008 標準では、USING キーワードが代わりに使用されます。

  ○SQL/2008 標準では、CURSOR 句はキーワード STRUCTURE で終了します。STRUCTURE は SQL Anywhere ではサポートされません。

例

次の例は、DESCRIBE 構造体の使用方法を示します。

```
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;
}
```

DESCRIBE 文 [Interactive SQL]

指定されたデータベースオブジェクトに関する情報を返します。

構文 1 - データベースオブジェクトの記述

DESCRIBE [[**INDEX FOR**] **TABLE** | **PROCEDURE**] [*owner.*]*object-name*

object-name :
table
| *view*
| *materialized view*
| *procedure*
| *function*

構文 2 - 現在の接続の記述

DESCRIBE CONNECTION

パラメーター

INDEX FOR 句　指定した *object-name* のインデックスを表示することを指定します。

TABLE 句　記述する *object-name* がテーブルまたはビューであることを示します。

PROCEDURE 句　*object-name* がプロシージャーまたはファンクションであることを示します。

備考

DESCRIBE TABLE を使用すると、指定したテーブルまたはビューのすべてのカラムが表示されます。DESCRIBE TABLE 文は 1 つのテーブルカラムにつき 1 つのローを返します。これには、次のものが含まれます。

- **カラム**　カラムの名前。
- **タイプ**　カラムに格納されているデータ型
- **NULL 入力可**　NULL が許可されているかどうか (1=許可、0=不許可)
- **プライマリキー**　カラムがプライマリキーにあるかどうか (1=ある、0=ない)

DESCRIBE INDEX FOR TABLE を使用すると、指定したテーブルのすべてのインデックスが表示されます。DESCRIBE TABLE 文は 1 つのインデックスにつき 1 つのローを返します。これには、次のものが含まれます。

- **インデックス名**　インデックスの名前。
- **カラム**　インデックス内のカラム。
- **ユニーク**　インデックスがユニークかどうか (1=ユニーク、0=ユニークではない)。
- **タイプ**　インデックスの型。可能な値は、「クラスタード」、「統計情報」、「ハッシュド」、「その他」です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DESCRIBE PROCEDURE を使用すると、指定したプロシージャーまたはファンクションが使用しているすべてのパラメーターが表示されます。DESCRIBE PROCEDURE 文は、1 つのパラメーターにつき 1 つのローを返します。これには、次のものが含まれます。

- **パラメーター**　パラメーターの名前。
- **タイプ**　パラメーターのユーザー型。
- **入力／出力**　パラメーターに渡すデータまたは返されるデータに関する情報。可能な値は、次のとおりです。
  - **入力**　パラメーターはプロシージャーに渡されますが、修正されません。
  - **出力**　プロシージャーはパラメーターの初期値を無視し、プロシージャーが返るときに値を設定します。
  - **入力／出力**　パラメーターはプロシージャーに渡され、プロシージャーはプロシージャーが返るときにパラメーターの値を設定します。
  - **結果**　パラメーターは結果セットを返します。
  - **戻り値**　パラメーターは宣言済みの戻り値を返します。

TABLE または PROCEDURE を指定していない場合 (たとえば、DESCRIBE *object-name*)、Interactive SQL はオブジェクトをテーブルと仮定します。ただし、そのようなテーブルが存在しない場合、Interactive SQL はプロシージャーまたはファンクションとしてオブジェクトを記述しようとします。

構文 2 を使用して Interactive SQL が接続されているデータベースまたはデータベースサーバーに関する情報をリストします。次のプロパティが返されます。

- **データベース製品**　Interactive SQL が接続されているデータベース製品の名前とバージョン番号 (SQL Anywhere 12.0.1.3600 など)。
- **ホスト名**　データベースサーバーを実行しているコンピューターのネットワーク名。
- **ホストの TCP/IP アドレス**　データベースサーバーを実行しているコンピューターの IP アドレス。
- **ホストのオペレーティングシステム**　データベースサーバーを実行しているコンピューターで使用されるオペレーティングシステムの名前とバージョン番号。
- **サーバー名**　データベースサーバーの名前。
- **サーバーの TCP/IP ポート**　現在の接続でデータベースサーバーによって使用されるポート番号。
- **データベース名**　Interactive SQL が接続されているデータベースの名前。
- **データベースの文字セット**　データベースの CHAR カラムに使用される文字セット。

- **接続文字列**　データベースまたはデータベースサーバーとの接続に使用された接続文字列。3 つのアスタリスクはパスワードに置き換えます。

現在の接続に該当しないプロパティは省略されます。たとえば、共有メモリを使用してデータベースサーバーに接続する場合、TCP/IP ポートは省略されます。

パーミッション

なし

関連する動作

なし

参照

- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

Departments テーブル内のカラムを記述します。

```
DESCRIBE TABLE Departments;
```

この文の結果セットの例を示します。

カラム	タイプ	NULL 入力可	プライマリキー
DepartmentID	integer	0	1
DepartmentName	char (40)	0	0
DepartmentHeadID	integer	1	0

Customers テーブルのインデックスを表示します。

```
DESCRIBE INDEX FOR TABLE Customers;
```

この文の結果の例を示します。

Index Name	Columns	Unique	Type
IX_customer_name	Surname,GivenName	0	Clustered

DETACH TRACING 文

診断トレーシングセッションを終了します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

DETACH TRACING { WITH | WITHOUT } SAVE

パラメーター

WITH SAVE 句　WITH SAVE を指定すると、保存されていない診断データが診断テーブルに保存されます。

WITHOUT SAVE 句　保存されていないトレーシングデータを保存しない場合は、WITHOUT SAVE を指定します。

備考

プロファイルされているデータベースからこの文を実行すると、診断テーブルに対する診断情報の送信が停止されます。WITHOUT SAVE 句を指定しても、sa_save_trace_data システムプロシージャーを使用して後でデータを保存できます (トレーシングデータベースが実行中で、他のトレーシングセッションが開始されていない場合)。

現在のトレーシングレベルを参照するには、sa_diagnostic_tracing_level テーブルを参照します。

> **注意**
> トレーシング情報は、データベースのアンロードまたは再ロード操作の一環としてアンロード**されません**。トレーシング情報を別のデータベースに転送する場合は、sa_diagnostic_* テーブルの内容をコピーして、手動で転送してください。ただし、この方法はおすすめしません。

パーミッション

DBA 権限または PROFILE 権限が必要です。

関連する動作

なし

参照

- 「ATTACH TRACING 文」487 ページ
- 「REFRESH TRACING LEVEL 文」872 ページ
- 「診断トレーシング」『SQL Anywhere サーバー SQL の使用法』
- 「sa_diagnostic_tracing_level テーブル」1022 ページ
- 「sa_save_trace_data システムプロシージャー」1165 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

DISCONNECT 文 [ESQL] [Interactive SQL]

データベースへの現在の接続を切断します。

構文

DISCONNECT [*connection-name* | **CURRENT** | **ALL**]

connection-name :
identifier
| *string*
| *hostvar*

備考

DISCONNECT 文はデータベースサーバーとの接続を切断し、データベースサーバーが使用するすべてのリソースを解放します。切断しようとする接続が CONNECT 文上で指定されている場合、その名前を使って接続を指定できます。ALL を指定すると、すべてのデータベース環境へのアプリケーションの接続がすべて切断されます。CURRENT はデフォルトであり、現在の接続を切断します。

commit_on_exit オプションが On に設定されている場合は、データベース接続を閉じる前に Interactive SQL が自動的に COMMIT を実行します。このオプションが Off に設定されている場合は、Interactive SQL は暗黙の ROLLBACK を実行します。デフォルトでは、commit_on_exit オプションは On に設定されています。

この文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

パーミッション

なし

関連する動作

なし

参照

- 「DROP CONNECTION 文」709 ページ
- 「CONNECT 文 [ESQL] [Interactive SQL]」516 ページ
- 「SET CONNECTION statement [Interactive SQL] [ESQL]」906 ページ
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　DISCONNECT は、SQL/2008 標準のオプションの SQL 言語機能 F771 を構成します。DISCONNECT をパラメーターなしで指定する機能は、ベンダー拡張です。commit_on_exit オプションはベンダー拡張です。

例

次の文は、Embedded SQL 内の DISCONNECT の使用法を示します。

```
EXEC SQL DISCONNECT :conn_name
```

次の文は、DISCONNECT を使用して、Interactive SQL との接続をすべて切断する方法を示します。

```
DISCONNECT ALL;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DROP CONNECTION 文

データベースへのユーザーの接続を削除します。

構文

DROP CONNECTION *connection-id*

備考

DROP CONNECTION 文は、データベースへの接続を削除してデータベースからユーザーを切断します。

connection-id パラメーターは整数の定数です。sa_conn_info システムプロシージャーを使用して、*connection-id* を取得できます。

この文は、プロシージャー、トリガー、イベント、またはバッチではサポートされていません。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「CONNECT 文 [ESQL] [Interactive SQL]」516 ページ
- 「sa_conn_info システムプロシージャー」1055 ページ
- 「例外ハンドラー」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次のプロシージャーは、接続番号で識別される接続を削除します。プロシージャー内から DROP CONNECTION 文を実行するときには、次の例に示すように、EXECUTE IMMEDIATE 文を使用して実行してください。

```
CREATE PROCEDURE drop_connection_by_id( IN conn_number INTEGER )
 BEGIN
   EXECUTE IMMEDIATE 'DROP CONNECTION ' || conn_number;
 END;
```

次の文は、ID 番号 4 の接続を削除します。

```
DROP CONNECTION 4;
```

DROP DATABASE 文

データベースに関連付けられているすべてのデータベースファイルを削除します。

構文

DROP DATABASE *database-name* [**KEY** *key*]

備考

DROP DATABASE 文は、関連するすべてのデータベースファイルをディスクから物理的に削除します。データベースファイルが存在しない場合、または起動に適した状態にない場合は、エラーが生成されます。

DROP DATABASE は、ストアドプロシージャー、トリガー、イベント、またはバッチに使用できません。

パーミッション

必要なパーミッションは、データベースサーバーの -gu オプションを使用して設定します。デフォルトの設定では、DBA 権限を必要とします。

使用中のデータベースは削除できません。

強力に暗号化されたデータベースを削除する場合は、キーを指定してください。

Windows Mobile ではサポートされません。

関連する動作

ディスクからデータベースファイルを削除するだけでなく、関連のトランザクションログファイルやトランザクションログミラーファイルも削除されます。

参照

- 「CREATE DATABASE 文」521 ページ
- 「DatabaseKey (DBKEY) 接続パラメーター」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

C:¥temp ディレクトリにあるデータベース *temp.db* を削除します。

```
DROP DATABASE 'c:¥temp¥temp.db';
```

DROP DATATYPE 文

データベースからデータ型を削除します。

構文

DROP DATATYPE *datatype-name*

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

DROP DATATYPE よりも DROP DOMAIN を使用することをおすすめします。これは、DROP DOMAIN が SQL/2008 標準で使用されている構文であるためです。システム定義のデータ型 (MONEY、UNIQUEIDENTIFIERSTR など) をデータベースから削除することはできません。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP DATATYPE 文を実行できます。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

参照

- 「DROP DOMAIN 文」712 ページ
- 「CREATE DOMAIN 文」533 ページ
- 「ALTER DOMAIN 文」416 ページ

標準と互換性

- **SQL/2008** ドメインのサポートは、SQL/2008 標準のオプションの SQL 言語機能 F251 です。DROP DATATYPE 文はベンダー拡張です。

例

データベースから MyDatatype を削除します。このデータ型が存在しない場合は、エラーが返されます。

```
DROP DATATYPE MyDatatype;
```

DROP DBSPACE 文

データベースから DB 領域を削除します。

構文

DROP DBSPACE *dbspace-name*

備考

DB 領域を削除するには、その前に DB 領域に含まれるすべてのテーブルを削除する必要があります。DROP DBSPACE 文を使用して、SYSTEM、TEMPORARY、TEMP、TRANSLOG、TRANSLOGMIRROR の事前定義の DB 領域を削除することはできません。

DROP DBSPACE 文は、他の接続で現在使用中のオブジェクトに影響を及ぼす場合は実行できません。

パーミッション

オブジェクトの所有者であるか DBA 権限を持っている必要があります。また、このデータベースに他の接続がないことが必要です。

関連する動作

オートコミット。暗黙的なチェックポイントが生じます。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

参照

- 「CREATE DBSPACE 文」528 ページ
- 「ALTER DBSPACE 文」414 ページ
- 「DB 領域の削除」『SQL Anywhere サーバー データベース管理』
- 「事前定義の DB 領域」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

データベースから MyDBSpace を削除します。この DB 領域が存在しない場合は、エラーが返されます。

```
DROP DBSPACE MyDBSpace;
```

DROP DOMAIN 文

データベースからドメインを削除します。

構文

DROP DOMAIN *domain-name*

備考

データ型をテーブルカラム、プロシージャーの引数、または関数の引数に使用する場合、DROP DOMAIN は許可されません。データ型を削除するには、ドメインを使用して定義されているすべてのカラムのデータ型を変更します。DROP DATATYPE よりも DROP DOMAIN を使用することをおすすめします。これは、DROP DOMAIN が SQL/2008 標準で使用されている構文であるためです。システム定義のデータ型 (MONEY、UNIQUEIDENTIFIERSTR など) をデータベースから削除することはできません。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP DOMAIN 文を実行できます。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

参照

- 「CREATE DOMAIN 文」533 ページ
- 「ALTER DOMAIN 文」416 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** ドメインのサポートは、SQL/2008 標準のオプションの SQL 言語機能 F251 です。

例

データベースからドメイン MyDomain を削除します。このドメインが存在しない場合は、エラーが返されます。

```
DROP DOMAIN MyDomain;
```

DROP EVENT 文

データベースからイベントを削除します。

構文

DROP EVENT [**IF EXISTS**] [*owner*.]*event-name*

備考

IF EXISTS 句は、存在しないイベントを DROP EVENT 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP EVENT 文を実行できます。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

参照

- 「CREATE EVENT 文」540 ページ
- 「ALTER EVENT 文」417 ページ
- 「TRIGGER EVENT 文」960 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

データベースから MyEvent を削除します。このイベントが存在しない場合は、エラーが返されます。

```
DROP EVENT MyEvent;
```

DROP EXTERNLOGIN 文

SQL Anywhere カタログから外部ログインを削除します。

構文

DROP EXTERNLOGIN *login-name* **TO** *remote-server*

パラメーター

DROP 句　ローカルユーザーログイン名を指定します。

TO 句　リモートサーバー名を指定します。このサーバーのローカルユーザーの代替ログイン名とパスワードが、削除される外部ログインです。

備考

DROP EXTERNLOGIN は、SQL Anywhere カタログから外部ログインを削除します。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「CREATE EXTERNLOGIN 文」548 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

```
DROP EXTERNLOGIN DBA TO sybase1;
```

DROP FUNCTION 文

データベースから関数を削除します。

構文

DROP FUNCTION [**IF EXISTS**] [*owner*.]*function-name*

備考

IF EXISTS 句は、存在しない関数を DROP FUNCTION 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

DROP FUNCTION 文は、他の接続で現在使用中のオブジェクトに影響を及ぼす場合は実行できません。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP FUNCTION 文を実行できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

参照

- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [外部呼び出し]」550 ページ
- 「CREATE FUNCTION 文 [Web サービス]」555 ページ
- 「ALTER FUNCTION 文」421 ページ

標準と互換性

- **SQL/2008** コア機能。IF EXISTS 句はベンダー拡張です。

例

データベースから MyFunction を削除します。この関数が存在しない場合は、エラーが返されます。

```
DROP FUNCTION MyFunction;
```

DROP INDEX 文

データベースからインデックスを削除します。

構文

DROP INDEX [**IF EXISTS**] { [[*owner*.]*table-name*.]*index-name* | [[*owner*.]*materialized-view-name*.]*index-name* }

備考

IF EXISTS 句は、存在しないインデックスを DROP INDEX 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

IF EXISTS 句を指定し、指定のテーブルが見つからない場合、エラーが返されます。

DROP INDEX 文は、他の接続で現在使用中のオブジェクトに影響を及ぼす場合は実行できません。

パーミッション

テーブルに対する REFERENCES パーミッションを持つユーザーは、DROP INDEX を実行できます。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、DROP INDEX 文は実行できません。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。DROP INDEX 文は、現在の接続のすべてのカーソルを閉じます。

ローカルテンポラリテーブルのインデックスを削除するために DROP INDEX 文を使用すると、インデックスが見つからないことを示すエラーが返されます。ローカルテンポラリテーブルの削除には、DROP TABLE 文を使用してください。ローカルテンポラリテーブル上のインデックスは、ローカルテンポラリテーブルがスコープ外になったときに自動的に削除されます。

参照

- 「CREATE INDEX 文」567 ページ
- 「ALTER INDEX 文」422 ページ
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

データベースから MyIndex を削除します。このインデックスが存在しない場合は、エラーが返されます。

```
DROP INDEX MyIndex;
```

DROP LOGIN POLICY 文

ログインポリシーを削除します。

構文

DROP LOGIN POLICY *policy-name*

パラメーター

policy-name　ログインポリシーの名前。

備考

ユーザーに割り当てられたログインポリシーを削除すると、文が失敗します。ルートログインポリシーは削除できません。ALTER USER 文を使用してユーザーのポリシー割り当てを変更します。

パーミッション

DBA 権限。

関連する動作

なし

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「ALTER LOGIN POLICY 文」424 ページ
- 「ALTER USER 文」482 ページ
- 「COMMENT 文」512 ページ
- 「CREATE LOGIN POLICY 文」573 ページ
- 「CREATE USER 文」677 ページ
- 「DROP USER 文」735 ページ
- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』
- 「ログインポリシーの削除」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、ログインポリシー Test11 を作成し、その後、削除します。

```
CREATE LOGIN POLICY Test11;
DROP LOGIN POLICY Test11;
```

DROP MATERIALIZED VIEW 文

データベースからマテリアライズドビューを削除します。

構文

DROP MATERIALIZED VIEW [**IF EXISTS**] [*owner*.]*materialized-view-name*

備考

テーブル内のすべてのデータは、削除プロセスの一部として自動的に削除されます。マテリアライズドビューのすべてのインデックスとキーも削除されます。

IF EXISTS 句は、存在しないマテリアライズドビューを DROP MATERIALIZED VIEW 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

他の接続で現在使用中のオブジェクトに対して DROP MATERIALIZED VIEW 文を実行することはできません。

DROP MATERIALIZED VIEW 文を実行すると、すべての通常の従属ビューのステータスが INVALID に変わります。マテリアライズドビューを削除する前にビューの依存関係を判断するには、sa_dependent_views システムプロシージャーを使用します。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP MATERIALIZED VIEW 文を実行できます。

関連する動作

オートコミット。マテリアライズドビューにデータが移植されている場合、DROP MATERIALIZED VIEW は自動チェックポイントをトリガーします。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。現在接続しているすべてのカーソルを閉じます。

ビューを削除すると、すべてのプロシージャーとトリガーがメモリからアンロードされます。これにより、削除されたビューを参照するプロシージャーやトリガーは、そのビューが存在しないことを反映します。ビューの削除や作成を頻繁に行うと、プロシージャーやトリガーのアンロードとロードによってパフォーマンスが低下することがあります。

参照

- 「CREATE MATERIALIZED VIEW 文」576 ページ
- 「ALTER MATERIALIZED VIEW 文」427 ページ
- 「REFRESH MATERIALIZED VIEW 文」867 ページ
- 「高度：マテリアライズドビューのステータスとプロパティ」『SQL Anywhere サーバー SQL の使用法』
- 「sa_dependent_views システムプロシージャー」1070 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

データベースから MyMaterializedView を削除します。このマテリアライズドビューが存在しない場合は、エラーが返されます。

```
DROP MATERIALIZED VIEW MyMaterializedView;
```

DROP MESSAGE 文

データベースからメッセージを削除します。

構文

DROP MESSAGE *msgnum*

備考

なし

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP MESSAGE 文を実行できます。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「PRINT 文 [T-SQL]」859 ページ
- 「CREATE MESSAGE 文 [T-SQL]」578 ページ
- 「ISYSUSERMESSAGE システムテーブル」1007 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Transact-SQL**　DROP MESSAGE には、Adaptive Server Enterprise の sp_dropmessage() システムプロシージャーによって提供される機能が用意されています。

例

データベースから MyMessage を削除します。このメッセージが存在しない場合は、エラーが返されます。

```
DROP MESSAGE MyMessage;
```

DROP MIRROR SERVER 文

> **注意**
> 読み込み専用のスケールアウトとデータベースミラーリングには、それぞれ別のライセンスが必要です。「別途ライセンスが必要なコンポーネント」『SQL Anywhere 12 紹介』を参照してください。

ミラーサーバーを削除します。

構文

DROP MIRROR SERVER *mirror-server-name*

備考

指定したミラーサーバーをデータベースから削除します。

パーミッション

DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「データベースミラーリング」『SQL Anywhere サーバー データベース管理』
- 「CREATE MIRROR SERVER 文」579 ページ
- 「ALTER MIRROR SERVER 文」430 ページ
- 「COMMENT 文」512 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、scaleout_server_root という名前のミラーサーバーをデータベースから削除します。

```
DROP MIRROR SERVER scaleout_server_root;
```

DROP PROCEDURE 文

データベースからプロシージャーを削除します。

構文

DROP PROCEDURE [IF EXISTS] [*owner.***]***procedure-name*

備考

IF EXISTS 句は、存在しないプロシージャーを DROP PROCEDURE 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

DROP PROCEDURE 文は、他の接続で現在使用中のオブジェクトに影響を及ぼす場合は実行できません。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP PROCEDURE 文を実行できます。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

参照

- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [外部呼び出し]」583 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「ALTER PROCEDURE 文」433 ページ

標準と互換性

- **SQL/2008** コア機能。IF EXISTS 句はベンダー拡張です。

例

データベースから MyProcedure を削除します。このプロシージャーが存在しない場合は、エラーが返されます。

```
DROP PROCEDURE MyProcedure;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DROP PUBLICATION 文 [Mobile Link] [SQL Remote]

パブリケーションを削除します。

構文

DROP PUBLICATION [**IF EXISTS**] [*owner*.]*publication-name*

owner, *publication-name* : *identifier*

備考

この文は、Mobile Link と SQL Remote にのみ適用されます。

Mobile Link では、パブリケーションが SQL Anywhere リモートデータベース内の同期データを識別します。SQL Remote では、統合データベース内とリモートデータベース内の両方のレプリケートされたデータがパブリケーションによって識別されます。

IF EXISTS 句は、存在しないパブリケーションを DROP PUBLICATION 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

パーミッション

DBA 権限を持っているか、パブリケーションの所有者であることが必要です。文中で参照されるすべてのテーブルに対する排他アクセスが必要です。

関連する動作

オートコミット。パブリケーションに対するサブスクリプションがすべて削除されます。

参照

- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」435 ページ
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」610 ページ
- SQL Anywhere Mobile Link クライアント： 「パブリケーション」『Mobile Link クライアント管理』
- Ultra Light Mobile Link クライアント： 「DROP PUBLICATION 文 [Ultra Light]」『Ultra Light データベース管理とリファレンス』
- SQL Anywhere Mobile Link クライアント： 「パブリケーションの削除」『Mobile Link クライアント管理』
- SQL Remote： 「パブリケーションの削除」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、pub_contact パブリケーションを削除します。

```
DROP PUBLICATION pub_contact;
```

DROP REMOTE MESSAGE TYPE 文 [SQL Remote]

データベースからメッセージタイプの定義を削除します。

構文

DROP REMOTE MESSAGE TYPE *message-system*

message-system :
FILE
| **FTP**
| **SMTP**

備考

この文は、データベースからメッセージタイプを削除します。

パーミッション

DBA 権限。および、対象のタイプに対する REMOTE または CONSOLIDATE パーミッションを持つユーザーが存在しないこと。

関連する動作

オートコミット。

参照

- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」613 ページ
- 「SQL Remote メッセージシステム」『SQL Remote』
- 「メッセージタイプの削除」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、データベースから FILE メッセージタイプを削除します。

```
DROP REMOTE MESSAGE TYPE file;
```

DROP SEQUENCE 文

シーケンスを削除します。

構文

DROP SEQUENCE [*owner*.] *sequence-name*

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

指定したシーケンスが見つからない場合は、エラーメッセージが返されます。シーケンスを削除すると、そのシーケンス名のすべての同義語がデータベースサーバーによって自動的に削除されます。

パーミッション

DBA 権限を持っているか、または、シーケンスの所有者であり、かつ RESOURCE 権限を持っている必要があります。

関連する動作

なし

参照

- 「ユニークな値を生成するためのシーケンスの使用」『SQL Anywhere サーバー SQL の使用法』
- 「ALTER SEQUENCE 文」437 ページ
- 「CREATE SEQUENCE 文」616 ページ

標準と互換性

- **SQL/2008**　シーケンスは、SQL/2008 のオプションの言語機能 T176 を構成します。

例

次の例では、Test という名前のシーケンスを削除します。

```
DROP SEQUENCE Test;
```

DROP SERVER 文

SQL Anywhere カタログからリモートサーバーを削除します。

構文

DROP SERVER *server-name*

備考

DROP SERVER は、SQL Anywhere カタログからリモートサーバーを削除します。この文を成功させるには、リモートサーバーに定義されたプロキシテーブルをすべて削除しておきます。

パーミッション

DBA 権限。

Windows Mobile ではサポートされません。

関連する動作

オートコミット。

参照

- 「CREATE SERVER 文」618 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

```
DROP SERVER ase_prod;
```

DROP SERVICE 文

Web サービスを削除します。

構文

DROP SERVICE *service-name*

備考

この文は、ISYSWEBSERVICE システムテーブルに表示されている Web サービスを削除します。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「CREATE SERVICE 文 [HTTP Web サービス]」622 ページ
- 「CREATE SERVICE 文 [SOAP Web サービス]」628 ページ
- 「ALTER SERVICE 文 [HTTP Web サービス]」441 ページ
- 「ALTER SERVICE 文 [SOAP Web サービス]」446 ページ
- 「ISYSWEBSERVICE システムテーブル」1008 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の SQL 文は、**WebServiceTable** という Web サービスを削除する方法を示します。

```
DROP SERVICE WebServiceTable;
```

DROP SPATIAL REFERENCE SYSTEM 文

空間参照系を削除します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

DROP SPATIAL REFERENCE SYSTEM [**IF EXISTS**] *name*

備考

IF EXISTS 句は、存在しない空間参照系を DROP SPATIAL REFERENCE SYSTEM 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

パーミッション

DBA 権限が必要です。

関連する動作

なし

参照

- 「CREATE SPATIAL REFERENCE SYSTEM 文」635 ページ
- 「ALTER SPATIAL REFERENCE SYSTEM 文」453 ページ
- 「空間データの使用」『SQL Anywhere サーバー 空間データサポート』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例では、TEST という名前の空間参照系を削除します。

```
DROP SPATIAL REFERENCE SYSTEM Test;
```

DROP SPATIAL UNIT OF MEASURE 文

空間測定単位を削除します。

構文

DROP SPATIAL UNIT OF MEASURE [**IF EXISTS**] *identifier*

備考

IF EXISTS 句は、存在しない空間測定単位を DROP SPATIAL UNIT OF MEASURE 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

パーミッション

DBA 権限を持つか、SYS_SPATIAL_ADMIN_ROLE のメンバーである必要があります。

関連する動作

なし

参照

- 「CREATE SPATIAL UNIT OF MEASURE 文」642 ページ
- 「空間データの使用」『SQL Anywhere サーバー 空間データサポート』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例では、TEST という名前の空間測定単位を削除します。

```
DROP SPATIAL UNIT OF MEASURE Test;
```

DROP STATEMENT 文 [ESQL]

文のリソースを解放します。

構文

DROP STATEMENT [*owner*.]*statement-name*

statement-name :
identifier
| *hostvar*

備考

DROP STATEMENT 文は、指定した文によって使われているリソースを解放します。これらのリソースは、PREPARE 文を実行して割り付けられます。通常、データベース接続が解放されるまで、リソースは解放されません。

パーミッション

文を準備しておいてください。

関連する動作

なし

参照

- 「PREPARE 文 [ESQL]」856 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。SQL/2008 標準では、この機能はオプションの SQL 言語機能 B032、"Extended dynamic SQL" の一部である、DEALLOCATE PREPARE 文 によって提供されます。

例

次は、DROP STATEMENT の使用例です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
EXEC SQL DROP STATEMENT S1;
EXEC SQL DROP STATEMENT :stmt;
```

DROP STATISTICS 文

指定したカラムのすべての統計情報を消去します。

構文

DROP STATISTICS [**ON**] [*owner.*]*object-name* [(*column-list*)]

object-name :
table-name
| *materialized-view-name*
| *temp-table-name*

備考

SQL Anywhere オプティマイザーは、カラムの統計情報を基にして、各文の実行に最適な方式を判別します。SQL Anywhere は、それらの統計を自動的に収集、更新します。カラムの統計情報は、データベースのシステムテーブル SYSCOLSTAT に永久的に格納されます。ある文の処理中に収集されたカラムの統計情報は、以降の文の効率的な実行方法を見いだすときに使用できます。

場合によっては、カラムの統計情報が不正確になったり、関連統計情報が使用不能になったりすることがあります。このような状況がもっとも発生しやすいのは、大量のデータが追加、更新、または削除されてから実行されたクエリが少ない場合です。

DROP STATISTICS 文は、システムテーブル ISYSCOLSTAT から、指定されたカラムに関する内部統計データをすべて削除します。これは強力な処理で、オプティマイザーは必要な統計情報にアクセスできなくなります。これらの統計情報がない場合、オプティマイザーは非効率的なデータアクセスプランを生成し、データベースパフォーマンスの低下を招くことがあります。

DROP STATISTICS 文には、実行対象のテーブルに排他ロックをかける必要があります。つまり、テーブルを参照するその他すべての接続がコミットまたはロールバックされるか、テーブルを参照している開いたカーソルをすべて閉じるまで、文の実行は進行しません。

この文は、問題を追究するときや、元のデータとは大幅に異なるデータをデータベースに再ロードする場合のみ使用してください。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「CREATE STATISTICS 文」643 ページ
- 「オプティマイザーの推定と統計」『SQL Anywhere サーバー SQL の使用法』
- 「ISYSCOLSTAT システムテーブル」995 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

DROP SUBSCRIPTION 文 [SQL Remote]

パブリケーションからユーザーのサブスクリプションを削除します。

構文

DROP SUBSCRIPTION TO *publication-name* [**(** *subscription-value* **)**]
 FOR *subscriber-id*, ...

subscription-value : *string*

subscriber-id : *string*

パラメーター

publication-name　このパブリケーション名でユーザーのサブスクリプションが作成されます。パブリケーションの所有者を含めることもできます。

subscription-value　パブリケーションのサブスクリプション式と照合される文字列。ユーザーは単一のパブリケーションに対して複数のサブスクリプションを持つことがあるので、この値は必須です。

subscriber-id　パブリケーションに対するサブスクライバーのユーザー ID。

備考

現在のデータベースのパブリケーションに対するユーザー ID の SQL Remote サブスクリプションを削除します。今後パブリケーションのデータが変更されても、ユーザー ID は更新を受け付けません。

SQL Remote では、パブリケーションとサブスクリプションは双方向の関係です。統合データベース上のパブリケーションに対するリモートユーザーのサブスクリプションを削除する場合、リモートデータベースでも統合データベースのサブスクリプションを削除してください。これは、リモートデータベースに対する更新が統合データベースに送信されるのを防ぐためです。

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「CREATE SUBSCRIPTION 文 [SQL Remote]」645 ページ
- 「STOP SUBSCRIPTION 文 [SQL Remote]」950 ページ
- 「ISYSSUBSCRIPTION システムテーブル」1004 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、pub_contact publication に対するユーザー ID SamS のサブスクリプションを削除します。

```
DROP SUBSCRIPTION TO pub_contact
FOR SamS;
```

DROP SYNCHRONIZATION PROFILE 文 [Mobile Link]

SQL Anywhere 同期プロファイルを削除します。

構文

DROP SYNCHRONIZATION PROFILE [IF EXISTS] *name*

パラメーター

name 削除する同期プロファイルの名前。

備考

同期プロファイルは、同期の制御に使用できる同期オプションの名前付きコレクションです。IF EXISTS 句は、存在しない同期プロファイルを DROP SYNCHRONIZATION PROFILE 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link]」646 ページ
- 「ALTER SYNCHRONIZATION PROFILE 文 [Mobile Link]」457 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

DROP SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]

リモートデータベース内の同期サブスクリプションを削除します。

構文

DROP SYNCHRONIZATION SUBSCRIPTION {*subscription-name* |
TO *publication-name*
[**FOR** *ml-username*, ...]}

パラメーター

subscription-name 削除するサブスクリプションの名前を指定します。

TO 句 パブリケーション名を指定します。

FOR 句 1 人以上のユーザーを指定します。

この句を省略すると、パブリケーションに対するデフォルトの設定が削除されます。

パーミッション

DBA 権限が必要です。パブリケーションで参照されるすべてのテーブルへの排他アクセスが必要です。

関連する動作

オートコミット。

参照

- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」458 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」647 ページ
- 「ISYSSYNC システムテーブル」1005 ページ
- 「Mobile Link サブスクリプションの削除」『Mobile Link クライアント管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例では、mysub という名前のサブスクリプションを削除します。

```
DROP SYNCHRONIZATION SUBSCRIPTION mysub;
```

次の例では、ユーザー ml_user1 とパブリケーション sales_publication との間のサブスクリプションを削除します。

```
DROP SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR "ml_user1";
```

次の例では、FOR 句を省略し、パブリケーション sales_publication のデフォルトの設定を削除します。

```
DROP SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DROP SYNCHRONIZATION USER 文 [Mobile Link]

SQL Anywhere リモートデータベースから 1 人以上の同期ユーザーを削除します。

構文

DROP SYNCHRONIZATION USER *ml-username*, ...

ml-username : *identifier*

備考

Mobile Link リモートデータベースから、1 人以上の同期ユーザーを削除します。

パーミッション

DBA 権限と、ユーザーによってサブスクライブされるパブリケーションで参照されるすべてのテーブルへの排他アクセス。

関連する動作

ユーザーに関連するすべてのサブスクリプションも削除されます。

参照

- 「ALTER SYNCHRONIZATION USER 文 [Mobile Link]」461 ページ
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」650 ページ
- 「ISYSSYNC システムテーブル」1005 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

データベースから Mobile Link ユーザー ml_user1 を削除します。

```
DROP SYNCHRONIZATION USER ml_user1;
```

DROP TABLE 文

データベースからテーブルを削除します。

構文

DROP TABLE [IF EXISTS] [*owner*.]*table-name*

備考

テーブルを削除すると、テーブル内のすべてのデータが削除プロセスの一部として自動的に削除されます。テーブルのすべてのインデックスとキーも削除されます。

IF EXISTS 句は、存在しないテーブルを DROP TABLE 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

DROP TABLE 文は、他の接続で現在使用中のテーブルに影響を及ぼす場合は実行できません。テーブルに依存するマテリアライズドビューがある場合も、DROP TABLE 文を実行できません。

DROP TABLE 文を実行すると、すべての通常の従属ビューのステータスが INVALID に変わります。テーブルを削除する前にビューの依存関係を判断するには、sa_dependent_views システムプロシージャーを使用します。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP TABLE 文を実行できます。

グローバルテンポラリテーブルは、このテンポラリテーブルを参照したすべてのユーザーが切断されるまで削除できません。

関連する動作

オートコミット。DROP TABLE によって自動チェックポイントが生じる場合もあります。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。DROP TABLE 文を実行すると、現在の接続のすべてのカーソルが閉じられます。

DROP TABLE 文は、ローカルテンポラリテーブルを削除するために使用できます。

参照

- 「テーブルの削除」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE TABLE 文」651 ページ
- 「ALTER TABLE 文」462 ページ
- 「sa_dependent_views システムプロシージャー」1070 ページ

標準と互換性

- **SQL/2008** DROP TABLE は、SQL/2008 標準のコア機能です。IF EXISTS 句はベンダー拡張です。宣言されたローカルテンポラリテーブルを DROP TABLE 文で削除する機能はベンダー拡張です。

例

データベースから MyTable を削除します。このテーブルが存在しない場合は、エラーが返されます。

```
DROP TABLE MyTable;
```

データベースに MyTable が存在する場合は、削除します。このテーブルが存在しない場合でも、エラーは返されません。

```
DROP TABLE IF EXISTS MyTable;
```

DROP TEXT CONFIGURATION 文

テキスト設定オブジェクトを削除します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

DROP TEXT CONFIGURATION [*owner.*]*text-config-name*

備考

依存するテキストインデックスがあるテキスト設定オブジェクトを削除しようとすると、エラーになります。依存するテキストインデックスを削除してからテキスト設定オブジェクトを削除してください。

テキスト設定オブジェクトは、ISYSTEXTCONFIG システムテーブルに格納されています。

パーミッション

テキスト設定オブジェクトの所有者であるか、DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「DROP TEXT INDEX 文」733 ページ
- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「SYSTEXTCONFIG システムビュー」1320 ページ
- 「CREATE TEXT CONFIGURATION 文」666 ページ
- 「ALTER TEXT CONFIGURATION 文」476 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の各文は、mytextconfig というテキスト設定オブジェクトをそれぞれ作成および削除します。

```
CREATE TEXT CONFIGURATION mytextconfig FROM default_char;
DROP TEXT CONFIGURATION mytextconfig;
```

DROP TEXT INDEX 文

データベースからテキストインデックスを削除します。

構文

DROP TEXT INDEX *text-index-name*
ON [*owner.*]*table-name*

パラメーター

ON 句　この句は、テキストインデックスが構築されたテーブルを指定するときに使用します。

備考

依存するテキストインデックスを削除してから、テキスト設定オブジェクトを削除できます。

パーミッション

基本となるテーブルの所有者であるか、DBA 権限または REFERENCES パーミッションが必要です。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、この文は実行できません。

関連する動作

オートコミット。

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキストインデックスの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』
- 「SYSTEXTCONFIG システムビュー」1320 ページ
- 「CREATE TEXT INDEX 文」667 ページ
- 「ALTER TEXT INDEX 文」480 ページ
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の各文は、TextIdx というテキストインデックスをそれぞれ作成および削除します。

```
CREATE TEXT INDEX TextIdx ON MarketingInformation ( Description )
DROP TEXT INDEX TextIdx ON MarketingInformation;
```

DROP TRIGGER 文

データベースからトリガーを削除します。

構文

DROP TRIGGER [**IF EXISTS**] [*owner*.] [*table-name*.]*trigger-name*

備考

IF EXISTS 句は、存在しないデータベースオブジェクトをこの DROP 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

テーブルに対する ALTER パーミッションを持つユーザーは、DROP TRIGGER を実行できます。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。

参照

- 「CREATE TRIGGER 文」670 ページ
- 「ALTER TRIGGER 文」481 ページ
- 「ROLLBACK TRIGGER 文」893 ページ
- 「トリガーの削除」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　DROP TRIGGER は、SQL/2008 標準のオプションの SQL 言語機能 T211、"Basic trigger capability" の一部です。IF EXISTS 句はベンダー拡張です。

例

データベースから MyTrigger を削除します。このトリガーが存在しない場合は、エラーが返されます。

```
DROP TRIGGER MyTrigger;
```

DROP USER 文

ユーザーを削除します。

構文

DROP USER *user-name*

パラメーター

- ***user-name***　削除するユーザーの名前。

パーミッション

DBA 権限。

備考

ユーザーを削除すると、そのユーザーが所有するすべてのデータベースオブジェクト (テーブルやプロシージャーなど) とそのユーザーの外部ログインも削除されます。この文の実行中は、対象ユーザーはデータベースに接続できません。

関連する動作

なし

参照

- 「ALTER LOGIN POLICY 文」424 ページ
- 「ALTER USER 文」482 ページ
- 「COMMENT 文」512 ページ
- 「CREATE LOGIN POLICY 文」573 ページ
- 「CREATE USER 文」677 ページ
- 「DROP LOGIN POLICY 文」716 ページ
- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、データベースからユーザー SQLTester を削除します。

```
DROP USER SQLTester;
```

DROP VARIABLE 文

SQL 変数を削除します。

構文

DROP VARIABLE [**IF EXISTS**] *identifier*

備考

DROP VARIABLE 文は、CREATE VARIABLE 文を使って以前に作成した SQL 変数を削除します。データベース接続を解放すると、変数は自動的に削除されます。変数は大規模なオブジェクトのためによく使われます。したがって、使用後にこれらを削除したり、NULL に設定したりすると、相当量のリソース (主にディスク領域) が解放されることがあります。

IF EXISTS 句は、存在しないデータベースオブジェクトをこの DROP 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

パーミッション

なし

関連する動作

なし

参照

- 「CREATE VARIABLE 文」679 ページ
- 「SET 文」927 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DROP VIEW 文

データベースからビューを削除します。

構文

DROP VIEW [**IF EXISTS**] [*owner*.]*view-name*

備考

IF EXISTS 句は、存在しないビューを DROP VIEW 文が削除しようとしたときにエラーを返さないようにする場合に使用します。

DROP VIEW 文を実行すると、すべての通常の従属ビューのステータスが INVALID に変わります。 ビューを削除する前にビューの依存関係を判断するには、sa_dependent_views システムプロシージャーを使用します。

1 つ以上の INSTEAD OF トリガーがあるビューで DROP VIEW 文を実行すると、エラーが返されます。 トリガーを削除してからでないと、ビューを削除または変更できません。

パーミッション

オブジェクトを所有するユーザー、または DBA 権限を持つユーザーであれば、DROP VIEW 文を実行できます。

関連する動作

オートコミット。Interactive SQL で **[結果]** ウィンドウ枠の **[結果]** タブをクリアします。 DROP VIEW 文を実行すると、現在の接続のすべてのカーソルが閉じられます。

ビューを削除すると、すべてのプロシージャーとトリガーがメモリからアンロードされます。これにより、削除されたビューを参照するプロシージャーやトリガーに、そのビューが存在しないことが反映されます。 ビューの削除や作成を頻繁に行うと、プロシージャーやトリガーのアンロードとロードによってパフォーマンスが低下することがあります。

参照

- 「CREATE VIEW 文」681 ページ
- 「ALTER VIEW 文」484 ページ
- 「sa_dependent_views システムプロシージャー」1070 ページ

標準と互換性

- **SQL/2008** DROP VIEW は、SQL/2008 標準のコア機能です。IF EXISTS 句はベンダー拡張です。

例

データベースから MyView を削除します。このビューが存在しない場合は、エラーが返されます。

```
DROP VIEW MyView;
```

EXCEPT 文

2 つのクエリブロックの集合差を返します。

構文

[**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*。「一般的な SQL 構文要素」403 ページを参照してください。

except-query-block : *query-block*。「一般的な SQL 構文要素」403 ページを参照してください。

option-name : *identifier*

option-value :
hostvar (許可されたインジケーター)
| *string*
| *identifier*
| *number*

パラメーター

main-query-block SELECT 文またはクエリ式を構成するクエリブロック (ネストされている可能性あり)。

except-query-block SELECT 文またはクエリ式を構成するクエリブロック (ネストされている可能性あり)。

FOR XML 句 FOR XML 句の詳細については、「SELECT 文」895 ページを参照してください。

OPTION 句 この句は、文を実行するためのヒントを指定するときに使用します。次のヒントがサポートされます。

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- *option-name* = *option-value* クエリテキスト内の OPTION(isolation_level = ...) の指定は、クエリの独立性レベルを指定する他のいずれの手段よりも優先されることに注意してください。

備考

EXCEPT 文は、main-query-block 内のすべてのロー (except-query-block 内にも存在するローを除く) を返します。main-query-block での重複が、結果で重複として表示されないようにする場合は、EXCEPT または EXCEPT DISTINCT を指定します。それ以外の場合は、EXCEPT ALL を指定します。クエリブロックはネストできることに注意してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

EXCEPT 単独での使用は、EXCEPT DISTINCT と同じです。

main-query-block と *except-query-block* は和両立する必要があります。それぞれの SELECT リストには同数の項目が含まれており、かつ、各項目の型は比較可能である必要があります。2 つの SELECT リスト内の対応する項目が異なるデータ型の場合、SQL Anywhere は結果の中から対応するカラムのデータ型を選択し、各 *query-block* のカラムを自動的にそれぞれ変換します。

EXCEPT ALL では、集合差ではなく、バッグ差を実装します。たとえば、*main-query-block* 内に特定の値を持つ 5 つの (重複) ローがあり、*except-query-block* 内に同一の値を持つ 2 つの重複ローがある場合、EXCEPT ALL は 3 つのローを返します。

main-query-block 内に重複ローがない場合、EXCEPT の結果は EXCEPT ALL の結果と同じになります。

表示されるカラム名は最初の *query-block* に表示されるカラム名と同じであり、これらの名前は ORDER BY 句と一致する式の名前を判断するために使用されます。結果セットのカラム名をカスタマイズする別の方法として、共通のテーブル式 (WITH 句) を使用する方法があります。

パーミッション

各 *query-block* には SELECT パーミッションが必要です。

関連する動作

なし

参照

- 「INTERSECT 文」810 ページ
- 「UNION 文」964 ページ
- 「SELECT 文」895 ページ
- OPTION 句、SELECT 文902 ページ

標準と互換性

- **SQL/2008**　EXCEPT DISTINCT は、SQL/2008 標準のコア機能です。EXCEPT ALL は、オプションの SQL 言語機能 F304 を構成します。EXCEPT とともに DISTINCT キーワードを明示的に指定する方法は、SQL/2008 標準のオプションの SQL 言語機能 T551 です。EXCEPT とともに ORDER BY 句を指定する方法は、SQL 言語機能 F850 です。ORDER BY 句を含む *query-block* は、SQL/2008 機能 F851 の構成要素です。ロー制限句 (SELECT TOP または LIMIT) を含むクエリブロックは、コンテキストに応じて、オプションの SQL 言語機能 F857 または F858 を構成します。FOR XML 句と OPTION 句はベンダー拡張です。

- **Transact-SQL**　EXCEPT は、Adaptive Server Enterprise ではサポートされていません。ただし、SQL Anywhere でサポートされている Transact-SQL ダイアレクトでは、EXCEPT ALL と EXCEPT DISTINCT の両方を使用できます。

例

EXCEPT の使用例については、「集合演算子と NULL」『SQL Anywhere サーバー SQL の使用法』を参照してください。

EXECUTE IMMEDIATE 文 [SP]

動的に作成された文をプロシージャーの中から実行できるようにします。

構文 1

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** }

構文 2 - Transact-SQL

EXECUTE (*string-expression* **)**

パラメーター

WITH QUOTES 句 WITH QUOTES または WITH QUOTES ON を指定すると、文字列式にある二重引用符はすべて識別子を区切るものと見なされます。WITH QUOTES を指定しない場合、または WITH QUOTES OFF を指定する場合、文字列式内の二重引用符は、quoted_identifier オプションの現在の設定に応じて処理されます。

WITH QUOTES は、ストアドプロシージャーに渡されるオブジェクト名が、実行される文の一部として使用される場合に役立ちます。ただし、名前に二重引用符が必要な場合や、quoted_identifier オプションが Off に設定されているときにプロシージャーが呼び出される場合があります。

WITH ESCAPES 句 WITH ESCAPES OFF を指定すると、文字列式のエスケープシーケンス (¥n、¥x、¥¥ など) がすべて無視されます。たとえば、連続する 2 つの円記号は、1 つの円記号に変換されるのではなく、2 つの円記号として残ります。デフォルトの設定は、WITH ESCAPES ON です。

WITH ESCAPES OFF には、動的に作成された文が円記号を含むファイル名を参照する場合に、この文の実行を容易にするという用途があります。

コンテキストによっては、*string-expression* のエスケープシーケンスは、EXECUTE IMMEDIATE 文が実行される前に変換されます。たとえば、複合文は実行される前に解析され、WITH ESCAPES の設定にかかわらず、エスケープシーケンスがこの解析中に変換されます。これらのコンテキストでは、WITH ESCAPES OFF を指定すると、これ以上は変換されません。次に例を示します。

```
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
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

WITH BATCH 句　WITH BATCH 句を使用して、EXECUTE IMMEDIATE 文でバッチの実行を管理できます。WITH BATCH OFF を設定すると、プロシージャーの実行時に誤って SQL を挿入するのを防ぐことができます。

WITH BATCH ON は dbo が所有するプロシージャー以外ではデフォルトです。

WITH BATCH OFF が使用されると、*string-expression* で指定された文は 1 つの文になるはずです。

WITH RESULT SET 句　WITH RESULT SET 句を使用すれば、サーバーでこの句を含むプロシージャーを正しく定義できます。WITH RESULT SET ON または WITH RESULT SET OFF を指定することは、プロシージャーが作成されるときだけではなく、プロシージャーが実行されるときの動作についても影響をもたらすことになります。デフォルトオプションは WITH RESULT SET OFF です。

WITH RESULT SET ON を指定することによって、EXECUTE IMMEDIATE 文で結果を返すことができます。この句を使用すると、この句を含むプロシージャーが結果セットを返すように指定されます。

備考

EXECUTE IMMEDIATE 文は、プロシージャーとトリガーの中から実行できる文の種類を増やします。これを使って、プロシージャーに渡されたパラメーターを使って作成される文のような動的に作成される文を実行できます。

文中のリテラル文字列は一重引用符で囲んでください。文字列リテラルでは、複数行をスキャンできません。

EXECUTE IMMEDIATE によって実行される文では、グローバル変数だけが参照できます。

Transact-SQL のストアドプロシージャーとトリガーでは、構文 2 しか使用できません。

EXECUTE IMMEDIATE で実行される文には、キャッシュされたプランがありません。

パーミッション

なし

関連する動作

なし。ただし、文が関連する動作としてのオートコミットを伴うデータ定義文である場合、そのコミットが起こります。

参照

- 「quoted_identifier オプション」『SQL Anywhere サーバー データベース管理』
- 「プロシージャー、トリガー、ユーザー定義関数、バッチで使用される EXECUTE IMMEDIATE」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [外部呼び出し]」583 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「BEGIN 文」497 ページ
- 「EXECUTE 文 [ESQL]」742 ページ
- 「EXECUTE 文 [T-SQL]」744 ページ

標準と互換性

- **SQL/2008**　EXECUTE IMMEDIATE は、SQL/2008 標準のオプションの SQL 言語機能 B031、"Basic dynamic SQL" です。*execute-option* 構文はベンダー拡張です。SQL/2008 標準では、結果セットを返す EXECUTE IMMEDIATE の使用が禁止されています。

- **Transact-SQL**　構文 2 は、EXECUTE IMMEDIATE の Transact-SQL ダイアレクトの構文です。*execute-option* 構文は Adaptive Server Enterprise でサポートされていません。

例

次の文は、テーブル名がパラメーターとしてプロシージャーに提供されているテーブルを作成します。

```
CREATE PROCEDURE CreateTableProc(
               IN tablename char(30)
               )
BEGIN
  EXECUTE IMMEDIATE
  'CREATE TABLE ' || tablename ||
  ' ( column1 INT PRIMARY KEY)'
END;
```

プロシージャーを呼び出して、テーブル mytable を作成します。

```
CALL CreateTableProc( 'mytable' );
```

EXECUTE 文 [ESQL]

準備された SQL 文を実行します。

構文 1

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*

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

sqlda-name : *identifier*

into-sqlda-name : *identifier*

構文 2

EXECUTE IMMEDIATE *statement*

statement : *string* | *hostvar*

パラメーター

USING 句　SELECT 文または CALL 文から返される結果は、変数リストの変数の中、または指定した SQLDA が記述するプログラムデータ領域の中のいずれかに入ります。OUTPUT (select リストまたはパラメーター) 対ホスト変数リスト、または OUTPUT 対 SQLDA 記述子配列は、どちらも対応が 1 対 1 です。

INTO 句　EXECUTE INTO を INSERT 文と一緒に使用する場合、挿入されたローは 2 番目の記述子に返されます。たとえば、オートインクリメントプライマリキーを使用するとき、またはプライマリキーの値を生成する BEFORE INSERT トリガーを使用するとき、EXECUTE 文により、即座にローを再フェッチし、そのローに割り当てられているプライマリキーの値を決定するためのメカニズムが提供されます。オートインクリメントキーと一緒に @@identity を使用する場合にも同じ結果が得られます。

ARRAY 句　オプションの ARRAY 句を提供された INSERT 文と一緒に使用して、ワイド挿入ができます。ワイド挿入は複数のローを同時に挿入し、パフォーマンスを改善します。整数値は、挿入するローの数です。SQLDA には各エントリの変数 (ロー数 * カラム数) を入れます。最初のローは SQLDA の変数 0 から (ロー当たりのカラム数) -1 に入り、以後のローも同様です。

備考

EXECUTE 文を使用して SQL 文を作成できます。データベースから多数のローを返す SELECT 文または CALL 文にはカーソルを使用します。

INSERT、UPDATE、または DELETE の実行が成功した後、SQLCA (SQLCOUNT) の *sqlerrd*[*2*] フィールドに、演算によって影響を受けるローの数が入ります。

- **構文 1**　以前に作成された名前付きの動的文を実行します。要求についての情報を提供するホスト変数プレースホルダー (バインド変数) が動的文にある場合は、*sqlda-name* に C 変数を指定します。この C 変数は SQLDA へのポインターであり、SQLDA には動的文のすべてのバインド変数のための記述子を含めておきます。この方法以外に、バインド変数を *hostvar-list* で指定する方法もあります。
- **構文 2**　バインド変数または出力がない文を PREPARE して EXECUTE するための省略形です。文字列またはホスト変数の中にある SQL 文を即座に実行し、完了時に削除します。

パーミッション

実行される文に関するパーミッションがチェックされます。

関連する動作

なし

参照

- 「EXECUTE IMMEDIATE 文 [SP]」740 ページ
- 「PREPARE 文 [ESQL]」856 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「Embedded SQL でのカーソル」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　EXECUTE 文は、SQL/2008 標準のオプションの SQL 言語機能 B031、"Basic dynamic SQL" の一部です。INTO 句は、オプションの言語機能 B032、"Extended dynamic SQL" の一部です。ARRAY 句はベンダー拡張です。

  また、Embedded SQL でサポートされている EXECUTE IMMEDIATE 文も、オプションの SQL 言語機能 B031 の一部です。

例

DELETE を実行します。

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM Employees WHERE EmployeeID = 105';
```

準備された DELETE 文を実行します。

```
EXEC SQL PREPARE del_stmt FROM
'DELETE FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE del_stmt USING :employee_number;
```

準備されたクエリを実行します。

```
EXEC SQL PREPARE sel1 FROM
'SELECT Surname FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE sel1 USING :employee_number INTO :surname;
```

EXECUTE 文 [T-SQL]

構文 1 は、Adaptive Server Enterprise と互換性のある、CALL 文の代替としてプロシージャーを呼び出します。構文 2 は、Transact-SQL で準備された SQL 文を実行します。

構文 1

[**EXECUTE** | **EXEC**][*@return_status* =] [*creator*.]*procedure_name* [*argument*, ...]

argument :
[*@parameter-name* =] *expression*
| [*@parameter-name* =] *@variable* [*output*]

構文 2

EXECUTE (*string-expression* **)**

備考

構文 1 は、Transact-SQL との互換性を保つために実装されています。EXECUTE はストアドプロシージャーを呼び出します。オプションとしてプロシージャーパラメーターを指定し、出力され

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

た値を取得し、ステータス情報を返します。Watcom SQL では、CALL 文または EXECUTE IMMEDIATE 文を使用します。

構文 2 を使用すると、Transact-SQL のストアドプロシージャーとトリガーの中で動的文を実行できます。EXECUTE 文は、プロシージャーとトリガーの中から実行できる文の種類を増やします。これを使って、プロシージャーに渡されるパラメーターを使って作成された文のように、動的に作成された文を実行できます。文の中のリテラル文字列は、一重引用符で囲みます。また、文は 1 行にしてください。EXECUTE 文の構文 2 は、Transact-SQL との互換性を保つために実装されていますが、Transact-SQL または Watcom-SQL のバッチとプロシージャーにも使用できます。

Transact-SQL の EXECUTE 文には、結果セットを予期することを示す方法はありません。Transact-SQL プロシージャーが結果セットを返すことを示すには、一例として次のような情報を指定する方法があります。

```
IF 1 = 0
    SELECT 1 AS a
```

Transact-SQL のストアドプロシージャーとトリガーの中で文を実行することもできます。

パーミッション

プロシージャーの所有者であるか、そのプロシージャーに対する EXECUTE パーミッションまたは DBA 権限が必要です。

関連する動作

なし

参照

- 「CALL 文」503 ページ
- 「EXECUTE 文 [ESQL]」742 ページ
- 「EXECUTE IMMEDIATE 文 [SP]」740 ページ

標準と互換性

- **SQL/2008**　構文 1 はベンダー拡張です。構文 2 は、SQL/2008 標準のオプションの SQL 言語機能 B031、"Basic dynamic SQL" の EXECUTE IMMEDIATE 文と同じ機能を提供します。ただし、構文 2 の構文は、SQL/2008 標準の構文と異なります。

例

次のプロシージャーは、構文 1 を示します。

```
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;
```

次の文は、パラメーターに 23 の入力値を提供するプロシージャーを実行します。Open Client または JDBC アプリケーションから接続している場合は、クライアントのウィンドウに PRINT

メッセージが表示されます。ODBC または Embedded SQL アプリケーションから接続している場合、このメッセージはデータベースサーバーメッセージウィンドウに表示されます。

```
EXECUTE p1 23;
```

次の文はプロシージャーを実行する代替方法です。これは、パラメーターがいくつかあるときに便利です。

```
EXECUTE p1 @var = 23;
```

次の文は、パラメーターにデフォルト値を使用するプロシージャーを実行します。

```
EXECUTE p1;
```

次の文は、プロシージャーを実行し、リターン状況をチェックするために変数へのリターン値を保管します。

```
EXECUTE @status = p1 23;
```

EXIT 文 [Interactive SQL]

Interactive SQL を終了します。

構文

{ **EXIT** | **QUIT** | **BYE** } [*return-code*]

return-code : *number* | *connection-variable*

備考

この文は、Interactive SQL がウィンドウプログラムとして実行している場合には Interactive SQL ウィンドウを閉じ、コマンドプロンプト (バッチ) モードで実行している場合には Interactive SQL を完全に終了します。どちらの場合でも、データベース接続も閉じられます。commit_on_exit オプションが On に設定されている場合は、データベース接続を閉じる前に Interactive SQL が自動的に COMMIT を実行します。このオプションが Off に設定されている場合は、Interactive SQL は暗黙の ROLLBACK を実行します。デフォルトでは、commit_on_exit オプションは On に設定されています。

オプションのリターンコードをバッチファイルでチェックすると、Interactive SQL スクリプトファイル内の文の成功または失敗を確認できます。デフォルトのリターンコードは 0 です。

パーミッション

なし

関連する動作

この文は、オプション commit_on_exit が On (デフォルト) に設定されている場合は自動的にコミットを実行し、そうでない場合は暗黙のロールバックを実行します。

Windows オペレーティングシステムでは、オプションの戻り値を ERRORLEVEL として使用できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SET OPTION 文」913 ページ
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、テーブル T にローが存在する場合は Interactive SQL の戻り値を 1 に設定し、テーブル T にローがない場合は 0 に設定します。

```
CREATE VARIABLE rowCount INT;
CREATE VARIABLE retcode INT;
SELECT COUNT(*) INTO rowCount FROM T;
IF( rowCount > 0 ) THEN
   SET retcode = 1;
ELSE
   SET retcode = 0;
END IF;
EXIT retcode;
```

注意

次の文は指定できません。これは、EXIT が (SQL 文ではなく) Interactive SQL 文であり、Interactive SQL 文を他の SQL ブロック文に含めることはできないためです。

```
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;
```

EXPLAIN 文 [ESQL]

特定のカーソルに使用されている最適化方法のテキスト仕様を取得します。

構文

EXPLAIN PLAN FOR CURSOR *cursor-name*
{ **INTO** *hostvar* | **USING DESCRIPTOR** *sqlda-name* }

cursor-name : *identifier* | *hostvar*

sqlda-name : *identifier*

備考

EXPLAIN 文は、指定したカーソルに対する最適化方法のテキスト表現を検索します。事前にカーソルを宣言し開いておきます。

hostvar または *sqlda-name* 変数には、文字列型を使用してください。最適化文字列は、どのような順序でテーブルを検索するかを指定し、もしあればどのインデックスを検索に使用するかを指定します。

この文字列はクエリによっては長くなることがあり、次のようなフォーマットに従います。

```
table (index), table (index), ...
```

テーブルに相関名が付いている場合、相関名がテーブル名の代わりになります。テーブル名がリストに並ぶ順序は、データベースサーバーがそれらにアクセスする順序です。それぞれのテーブルの後には、カッコで囲んだインデックス名が置かれます。これは、テーブルへのアクセスに使用するインデックスです。インデックスを使わない場合 (テーブルを連続的にスキャンします)、英字 "seq" がインデックス名の代わりになります。特別の SQL SELECT 文が部分文字列を伴う場合、コロン (:) でそれぞれのサブクエリの最適化文字列が分割されます。これらのサブクエリセクションは、データベースサーバーがクエリを実行する順序で表示されます。

EXPLAIN 文が正常に実行された後で、SQLCA (SQLIOESTIMATE) の sqlerrd フィールドに、クエリのすべてのローをフェッチするのに必要な入出力操作の数の推定値が格納されます。

最適化文字列の例については、「パフォーマンスのモニタリングと診断のツール」『SQL Anywhere サーバー SQL の使用法』を参照してください。

パーミッション

指定するカーソルを事前に開いておいてください。

関連する動作

なし

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「PREPARE 文 [ESQL]」856 ページ
- 「FETCH 文 [ESQL] [SP]」749 ページ
- 「CLOSE 文 [ESQL] [SP]」510 ページ
- 「OPEN 文 [ESQL] [SP]」844 ページ
- 「Embedded SQL でのカーソル」『SQL Anywhere サーバー プログラミング』
- 「SQLCA (SQL Communication Area)」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、EXPLAIN の使用方法を示します。

```
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;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
EXEC SQL OPEN employee_cursor;
EXEC SQL EXPLAIN PLAN FOR CURSOR employee_cursor INTO :plan;
printf( "Optimization Strategy: '%s'.n", plan );
```

計画変数には以下の文字列が含まれます。

```
'Employees <seq>'
```

FETCH 文 [ESQL] [SP]

カーソルを特定のローに配置 (再配置) し、そのローの式の値を、ストアドプロシージャーまたはアプリケーション内からアクセスできる変数にコピーします。

構文 1 [SP]

FETCH [*cursor-position*] *cursor-name*
INTO *variable-list* [**FOR UPDATE**]

構文 2 [ESQL]

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 : インジケーター変数を含めることができる

variable-list : ストアドプロシージャー変数

sqlda-name : *identifier*

fetch-count : *integer* | *hostvar*

パラメーター

- **INTO 句**　INTO 句はオプションです。この句が指定されていない場合、FETCH 文はカーソルのみを配置します。*hostvar-list* は Embedded SQL でのみ使用されます。

- **カーソル位置**　オプションの位置パラメーターを指定して、カーソルを移動してからローをフェッチできます。指定しない場合、NEXT が使用されます。FETCH 文に配置パラメーターがあり、その場所が許可されたカーソルの範囲の外側である場合、SQLE_NOTFOUND の警告が表示され、SQLCOUNT フィールドには有効な位置からのオフセットが設定されます。

OPEN 文は、まず、先頭のローの前にカーソルを配置します。

- **NEXT 句**　デフォルトの位置付けは NEXT です。これはローをフェッチする前に、カーソルを 1 つ前方のローに進めます。
- **PRIOR 句**　ローをフェッチする前に、カーソルを 1 つ後方のローに戻します。
- **RELATIVE 句**　RELATIVE 配置を使用すると、フェッチする前にいずれかの方向へ指定した数のローだけカーソルを移動できます。正の数は前進を示し、負の数は後退を示します。そのため、NEXT は RELATIVE 1 と等しく、PRIOR は RELATIVE -1 と等しくなります。RELATIVE 0 は、このカーソル上の最後の FETCH 文と同じローを取り出します。
- **ABSOLUTE 句**　ABSOLUTE 配置パラメーターを使用すると、特定のローに移動できます。0 は先頭のローの前の位置を示します。

  1 は先頭のローを示し、残りのローがそれに続きます。負の数を使ってカーソルの最後からの絶対位置を指定します。-1 はカーソルの最後のローを示します。
- **FIRST 句**　ABSOLUTE 1 の省略形。
- **LAST 句**　ABSOLUTE -1 の省略形。

> **注意**
> DYNAMIC SCROLL カーソルに挿入や更新をいくつか行うと、カーソルの位置の問題が生じます。SELECT 文に ORDER BY 句を指定しないかぎり、データベースサーバーはカーソル内の予測可能な位置にはローを挿入しません。場合によっては、カーソルを閉じてもう一度開かないと、挿入したローが表示されないことがあります。
>
> これは、カーソルを開くためにテンポラリテーブルを作成する必要がある場合に起こります。
>
> UPDATE 文によって、カーソル内のローが移動することがあります。これは、既存のインデックスを使用する ORDER BY 句がカーソルに指定されている場合に発生します (テンポラリテーブルは作成されません)。

- **BLOCK 句**　クライアントアプリケーションは一度に複数のローをフェッチできます。これはブロックフェッチ、プリフェッチ、またはマルチローフェッチと呼ばれます。最初のフェッチによって、いくつかのローがデータベースサーバーから送り返されます。クライアントはこれらのローをバッファーに格納します。後に続くフェッチは、データベースサーバーへ新しい要求を行わないで、これらのバッファーからローを取り出します。

  BLOCK 句は Embedded SQL でのみ使用されます。この句は、クライアントとサーバーに、アプリケーションがフェッチできるローの個数に関する情報を与えます。0 という特別値は、要求をデータベースサーバーに送信し、シングルローを返すことを示します (ローブロックはありません)。BLOCK 句を指定すると、BLOCK 値を次にプリフェッチするときに含められるローの数が削減されます。プリフェッチされるローの数を増やすには、PrefetchRows 接続パラメーターを使用します。

  BLOCK 句を指定しない場合は、OPEN に指定した値が使用されます。

  FETCH RELATIVE 0 は常にローを再フェッチします。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カーソルのプリフェッチが無効な場合、BLOCK 句は無視され、ローは一度に 1 つずつフェッチされます。ARRAY も指定している場合、ARRAY で指定されたロー数がフェッチされます。

- **PURGE 句**　PURGE 句は Embedded SQL でのみ使用されます。この句によって、クライアントはすべてのローのバッファーをフラッシュし、次にフェッチ要求をデータベースサーバーに送信します。このフェッチ要求はローのブロックを返すことに注意してください。

- **FOR UPDATE 句**　FOR UPDATE 句は、フェッチされたローが続いて UPDATE WHERE CURRENT OF CURSOR 文によって更新されることを示します。この句は、データベースサーバーがローに対して意図的なロックを設定するようにします。ロックは、現在のトランザクションの終わりまで保持されます。

- **ARRAY 句**　ARRAY 句は Embedded SQL でのみ使用されます。この句を使うと、いわゆるワイドフェッチができるようになります。これは複数のローを同時に取り出し、パフォーマンスを改善します。

  Embedded SQL でワイドフェッチを使用するには、コードに次のような FETCH 文を含めます。

  ```
  EXEC SQL FETCH ... ARRAY nnn
  ```

  ARRAY *nnn* は FETCH 文の最後の項目です。フェッチ回数を示す *nnn* にはホスト変数も使用できます。SQLDA には nnn * (ローあたりのカラム数) 変数を入れてください。最初のローは SQLDA の変数 0 から (ロー当たりのカラム数) -1 に入り、以後のローも同様です。

備考

FETCH 文は指定したカーソルからローを 1 つ取り出します。事前にカーソルを開いておきます。

- **Embedded SQL での使用**　DECLARE CURSOR 文は、C ソースコード内の FETCH 文の前に置きます。また、OPEN 文を実行してから FETCH 文を実行します。ホスト変数をカーソル名の代わりに使用している場合、DECLARE 文は実際にコードを生成するため、FETCH 文の前に実行します。

  サーバーは SQLCOUNT にフェッチされたレコードの数を返し、エラーがない場合は 0 より大きい SQLCOUNT を常に返します。

  フェッチ時に SQLSTATE_NOTFOUND 警告が返される場合、SQLCA (SQLCOUNT) の *sqlerrd*[*2*] フィールドには、フェッチの試みが許可されるカーソル位置を超えたときのローの数が含まれます。ローが見つからなくても位置が有効な場合は、値は 0 です。たとえば、カーソル位置が最後のローのときに FETCH RELATIVE 1 を実行した場合です。カーソルの最後を超えてフェッチしようとした場合、値は正の数です。カーソルの先頭を超えてフェッチしようとした場合、値は負の数です。カーソルの最後を超えてフェッチしようとした場合、カーソルの位置は最後のローになります。カーソルの先頭より前をフェッチしようとした場合、カーソルの位置は先頭のローになります。

  FETCH 文の実行が成功した後、SQLCA (SQLIOCOUNT) の *sqlerrd*[*1*] フィールドはフェッチを実行するのに必要な入出力操作の数だけ増加します。このフィールドは、実際にはデータベース文が発行されるたびに増加します。

- **シングルローフェッチ**　SELECT 文の結果からの 1 つのローは、変数リストの変数の中に置かれます。SELECT リスト対ホスト変数リストは、1 対 1 に対応します。

- **マルチローフェッチ**　SELECT 文の結果からの 1 つまたは複数のローは、*variable-list* の変数の中、または指定した *sqlda-name* が記述するプログラムデータ領域の中のいずれかに置かれます。SELECT リスト対 *hostvar-list* または対 *sqlda-name* 記述子配列は、どちらも 1 対 1 で対応します。

パーミッション

カーソルを開いておいてください。また、ユーザーはカーソルの宣言内で参照されるテーブルの SELECT パーミッションが必要です。

関連する動作

プリフェッチが有効な場合、FETCH 文によって、複数のローがサーバーからクライアントに取得されるようになる場合があります。

参照

- 「SELECT 文」895 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「FOR 文」753 ページ
- 「PREPARE 文 [ESQL]」856 ページ
- 「OPEN 文 [ESQL] [SP]」844 ページ
- 「RESUME 文」881 ページ
- 「ロックの仕組み」『SQL Anywhere サーバー SQL の使用法』
- 「データのフェッチ」『SQL Anywhere サーバー プログラミング』
- 「一度に複数のローをフェッチする」『SQL Anywhere サーバー プログラミング』
- 「クエリ処理におけるワークテーブルの使用 (All-rows 最適化ゴールの使用)」『SQL Anywhere サーバー SQL の使用法』
- 「Embedded SQL でのカーソル」『SQL Anywhere サーバー プログラミング』
- 「プロシージャー、トリガー、ユーザー定義関数、バッチのカーソル」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　若干の例外がありますが、FETCH 文の構文 1 は SQL/2008 標準のコア機能です。NEXT 以外のスクロールオプションは、オプションの SQL 言語機能 F431、"Read-only scrollable cursors" を構成します。SQL Anywhere では、SQL/2008 標準に記載されているように、FETCH 文のオプションの FROM 句はサポートされていません。

  構文 2 はベンダー拡張です。

  FOR UPDATE、PURGE、ARRAY、BLOCK、USING [SQL] DESCRIPTOR 句はベンダー拡張です。

例

次は、Embedded SQL の例です。

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT EmployeeID, Surname FROM Employees;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

次は、プロシージャーの例です。

```
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
```

FOR 文

文リストの実行をカーソル内のローごとに 1 回繰り返します。

構文

[*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*

パラメーター

NO SCROLL 句　NOSCROLL として宣言されたカーソルは、FETCHNEXT と FETCHRELATIVE0 の各シーク操作を使用して、結果セットの前方移動に制限されます。

カーソルがローを出た後は、そのローに戻ることができないため、カーソルに sensitivity の制限はありません。NO SCROLL カーソルが要求されると、SQL Anywhere は最も効率的なカーソルである asensitive カーソルを提供します。

DYNAMIC SCROLL 句　DYNAMIC SCROLL はデフォルトカーソルタイプです。DYNAMIC SCROLL カーソルでは、FETCH 文のすべてのフォーマットを使用できます。

DYNAMIC SCROLL カーソルを要求されると、SQL Anywhere は asensitive カーソルを提供します。カーソルを使用する場合、効率と一貫性の間には常にトレードオフ関係があります。Asensitive カーソルを使用すると、パフォーマンスは向上しますが一貫性が低下します。

SCROLL 句　SCROLL として宣言されたカーソルは、FETCH 文のすべてのフォーマットを使用できます。SCROLL カーソルを要求されると、SQL Anywhere は value-sensitive カーソルを提供します。

SQL Anywhere は、結果セットのメンバーシップが保証されるような方法で value-sensitive カーソルを実行します。DYNAMIC SCROLL カーソルの方が効率的です。SCROLL カーソルの一貫した動作が必要でない場合は、DYNAMIC SCROLL カーソルを使用してください。

INSENSITIVE 句　INSENSITIVE として宣言されたカーソルは、存続期間中、値とメンバーシップが固定されます。SELECT 文の結果セットはカーソルが開かれると、実体化されます。INSENSITIVE カーソルからの FETCHING は、そのカーソルを開いた接続を含むどの接続からの INSERT、UPDATE、MERGE、PUT、DELETE 文の影響も受けません。

SENSITIVE 句　SENSITIVE として宣言されたカーソルは、結果セットのメンバーシップまたは値の変更に依存します。

FOR UPDATE 句　FOR UPDATE はデフォルトです。 ORDER BY 句なしの単一テーブルのクエリの場合、または ansi_update_constraints オプションが Off に設定されている場合、カーソルのデフォルトは FOR UPDATE です。ansi_update_constraints オプションが Cursors または Strict に設定されている場合、ORDER BY 句を含むクエリ上のカーソルのデフォルトは READ ONLY です。ただし、FOR UPDATE 句を使用して、カーソルを更新可能と明示的に示すこともできます。

FOR READ ONLY 句　FOR READ ONLY として宣言されたカーソルは、UPDATE (位置付け) 文、DELETE (位置付け) 文、または PUT 文には使用できません。ORDER BY 句またはジョインを使用してカーソルに対する更新を可能にするにはコストがかかるため、2 つ以上のテーブルのジョインを含むクエリに対するカーソルは READ ONLY であり、ansi_update_constraints データベースオプションが Off でないかぎり、更新可能にすることはできません。FOR UPDATE を指定したカーソル要求への応答では、SQL Anywhere は value-sensitive カーソルまたは sensitive カーソルを提供します。insensitive カーソルと asensitive カーソルは更新できません。

参照

- 「sensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「insensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「value-sensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「asensitive カーソル」『SQL Anywhere サーバー プログラミング』

備考

FOR 文は制御文です。これを使うと、カーソル内の各ローに対して一度だけ、SQL 文のリストを実行できます。FOR 文は、カーソルに対して DECLARE を持つ複合文と同じです。また、カーソルの結果セットの中に、それぞれのカラムに対する変数の DECLARE がある複合文と同じです。複合文の後には、カーソルからローカル変数の中へローを一度フェッチし、カーソル内で各ローに対して一度だけ *statement-list* を実行するループが続きます。

有効なカーソルタイプには、dynamic scroll (デフォルト)、scroll、no scroll、sensitive、insensitive があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

それぞれのローカル変数の名前とデータ型は、カーソル内で使用する *statement* から取り出されます。SELECT 文の場合、データ型は SELECT リスト内の式のデータ型です。名前は、SELECT リスト項目のエイリアスが存在する場合はエイリアスになります。そうでない場合は、カラムの名前となります。SELECT リスト項目のカラム参照が簡単でない場合は、エイリアスを指定します。CALL 文の場合、名前とデータ型はプロシージャー定義内の RUSULT 句から取ります。

LEAVE 文を使って、END FOR に続く最初の文から実行を再開できます。終了の *statement-label* を指定する場合は、開始の *statement-label* と一致させます。

FOR 文によって作成されるカーソルは暗黙的に WITH HOLD を使用して開かれるため、このカーソルは、ループ内で実行され、COMMIT を発生させる文によって閉じられません。

パーミッション

なし

関連する動作

なし

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「FETCH 文 [ESQL] [SP]」749 ページ
- 「 CONTINUE 文」520 ページ
- 「LOOP 文」832 ページ

標準と互換性

- **SQL/2008**　FOR 文は、オプションの SQL/2008 言語機能 P002、"Computational completeness" の一部です。FOR 文の USING 句はベンダー拡張です。DECLARE CURSOR 文では、カーソル感知性とカーソルスクロール動作のオプションの組み合わせと同様、*cursor-concurrency* の使用はベンダー拡張です。

例

次のフラグメントは、FOR ループの使い方を示します。

```
FOR names AS curs INSENSITIVE CURSOR FOR
SELECT Surname
FROM Employees
DO
  CALL search_for_name( Surname );
END FOR;
```

次のフラグメントは、FOR ループの使い方を示します。

```
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
```

次の例は、プロシージャー myproc の内部で使用している FOR ループを示します。このプロシージャーは、プロシージャーを呼び出したときに指定したソート順に従って、Employees テーブルから上位 10 名の従業員を返します (asc は昇順、desc は降順)。

```
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 SCROLL CURSOR USING @sql
   DO
      INSERT INTO temp_names( surnames ) VALUES( Surname );
      MESSAGE( @msg || Surname ) ;
   END FOR;
   SELECT * FROM temp_names;
END ;
```

asc を指定して myproc プロシージャーを呼び出すと (たとえば、CALL myproc('asc');)、次のような結果が返されます。

Surname
Ahmed
Barker
Barletta
Bertrand
Bigelow
Blaikie
Braun
Breault
Bucceri

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Surname
Butterfield

FORWARD TO 文

ネイティブ構文の SQL 文をリモートサーバーに送信します。

構文 1

FORWARD TO *server-name sql-statement*

構文 2

FORWARD TO [*server-name*]

パラメーター

server-name リモートサーバーの名前。

sql-statement リモートサーバーのネイティブ SQL 構文によるコマンド。コマンドまたは一連のコマンドは、中カッコ ({}) または一重引用符で囲まれます。

備考

FORWARD TO 文は、ユーザーがパススルー接続が必要なサーバーを指定できるようにします。この文は、次の 2 つの方法で使用できます。

- **構文 1** 1 つの文をリモートサーバーに送信します。
- **構文 2** SQL Anywhere をパススルーモードにして一連の文をリモートサーバーに送信します。それ以降の文はすべて直接リモートサーバーに渡されます。パススルーモードをオフにするには、*server-name* を指定しないで FORWARD TO を実行します。

  パススルーモードでリモートサーバーからエラーが返された場合は、FORWARD TO 文を実行してパススルーモードをオフにしてください。

ユーザーのために server-name への接続を確立する場合、データベースサーバーは次のいずれかを使用します。

- CREATE EXTERNLOGIN を使用するリモートログインエイリアスセット
- リモートログインエイリアスが設定されない場合は、SQL Anywhere との通信に使用される名前とパスワード

指定のサーバーに接続を確立できない場合は、その理由を示すメッセージがユーザーに返されます。

文が要求されたサーバーに渡されると、結果はすべて、クライアントプログラムで認識できる形式に変換されます。

> **注意**
> FORWARD TO 文はサーバーディレクティブであり、ストアドプロシージャー、トリガー、イベント、またはバッチ内では使用できません。

パーミッション

なし

関連する動作

リモート接続は、FORWARD TO セッションの間、AUTOCOMMIT (非連鎖) モードに設定されます。FORWARD TO 文の前に保留中であった作業はすべて自動的にコミットされます。

参照

- 「PASSTHROUGH 文 [SQL Remote]」855 ページ

例

次の例では、SQL 文をリモートサーバー RemoteASE に送信します。

```
FORWARD TO RemoteASE { SELECT * FROM titles };
```

次の例は、リモートサーバー aseprod でのパススルーセッションを示します。

```
FORWARD TO aseprod
  SELECT * FROM titles
     SELECT * FROM authors
FORWARD TO;
```

標準と互換性

- **SQL/2008**　ベンダー拡張。

FROM 句

DELETE、SELECT、または UPDATE 文に必要なデータベーステーブルまたはビューを指定します。SELECT 文内で使用される場合、FROM 句は MERGE 文または INSERT 文でも使用できます。

構文

FROM *table-expression*, ...

table-expression :
table-name
| *view-name*
| *procedure-name*
| *derived-table*
| *lateral-derived-table*
| *join-expression*
| **(** *table-expression*, ... **)**
| *openstring-expression*
| *apply-expression*

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| *contains-expression*
| *dml-derived-table*

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*, ... **)**]

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

update-statement
merge-statement

table-version-names :
OLD [**AS**] *correlation-name* [**FINAL** [**AS**] *correlation-name*]
| **FINAL** [**AS**] *correlation-name*

パラメーター

- ***table-name*** ベーステーブルまたはテンポラリテーブル。ユーザー ID を指定すると、他のユーザーが所有するテーブルを修飾できます。ユーザー ID を指定しないと、現在のユーザーの所属グループが所有するテーブルがデフォルトで検索されます。

- ***view-name*** クエリに含めるビューを指定します。テーブルの場合と同様に、ユーザー ID を指定して、他のユーザーが所有するビューを修飾できます。ユーザー ID を指定しないと、現在のユーザーの所属グループが所有するビューがデフォルトで検索されます。構文ではビューに対してテーブルヒントを指定できますが、このようなヒントの効果はありません。

- ***procedure-name*** 結果セットを返すストアドプロシージャーです。この句は SELECT 文の FROM 句にのみ適用されます。プロシージャーにパラメーターを指定しない場合でも、プロシージャー名の後のカッコは必要です。オプションのパラメーターの代わりに DEFAULT を指定できます。

  ストアドプロシージャーが複数の結果セットを返す場合、最初の結果セットだけが使用されます。

  WITH 句を指定すると、プロシージャーの結果セットにカラム名のエイリアスを指定できます。WITH 句を指定する場合、カラム数はプロシージャーの結果セットのカラム数と一致し、データ型はプロシージャーの結果セットのデータ型と互換性がある必要があります。WITH 句を指定しない場合、カラム名と型はプロシージャー定義で設定された名前と型です。次のクエリは、WITH 句の使用方法を示します。

  ```
  SELECT sp.ident, sp.quantity, Products.name
  FROM ShowCustomerProducts( 149 )
    WITH ( ident INT, description CHAR(20), quantity INT ) sp
  JOIN Products
  ON sp.ident = Products.ID;
  ```

  Embedded SQL アプリケーションの場合、RESULT 句を使用しないでプロシージャーを作成し、そのプロシージャーが変数結果セットを返す場合には、プロシージャーを参照する SELECT 文の DESCRIBE が失敗することがあります。DESCRIBE の失敗を防ぐには、予測される結果セットスキーマを記述する WITH 句を含めることをおすすめします。

- ***derived-table*** FROM 句の中で、テーブル名またはビュー名の代わりに SELECT 文を指定できます。このように使用する SELECT 文を派生テーブルと呼び、エイリアスを指定する必要があります。たとえば、次の文には、派生テーブル MyDerivedTable が含まれています。この派生テーブルは、Products テーブル内の製品を UnitPrice でランキングしています。

  ```
  SELECT TOP 3 *
      FROM ( SELECT Description,
                    Quantity,
                    UnitPrice,
                    RANK() OVER ( ORDER BY UnitPrice ASC )
  ```

```
            AS Rank
            FROM Products ) AS MyDerivedTable
ORDER BY Rank;
```

- ***lateral-derived-table*** 親の文のオブジェクトへの参照 (外部参照) が含まれている場合がある、派生テーブル、ストアドプロシージャー、またはジョインしたテーブル。FROM 句の外部参照を使用する場合は、ラテラル派生テーブルを使用します。

  外部参照を使用できるのは、FROM 句のラテラル派生テーブルの前のテーブルに対してだけです。たとえば、SELECT リストの項目に外部参照は使用できません。

  テーブルと外部参照は、カンマで区切ります。たとえば、次のクエリは有効です。

  ```
  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;
  ```

  LATERAL (*table-expression*) の指定は、LATERAL (SELECT * FROM *table-expression*) の指定と同等です。

- ***openstring-expression*** OPENSTRING 句は、ファイルまたは BLOB 内を問い合わせ、これらのソースの内容をローセットとして処理するときに指定します。テーブルやビューなどの定義された構造を問い合わせているわけではないため、この句を指定するときは、ファイルまたは BLOB のスキーマ情報も指定して結果セットを生成します。この句は SELECT 文の FROM 句に適用されます。UPDATE 文または DELETE 文ではサポートされません。

  ROWID 関数は、OPENSTRING 式で生成されたテーブルの結果セットでサポートされます。

  次に示す OPENSTRING 句のサブ句とパラメーターは、ファイルと BLOB 内のデータを定義し、問い合わせるために使用します。

  FILE 句と VALUE 句 FILE 句は、問い合わせるファイルを指定するときに使用します。VALUE 句は、問い合わせる BLOB 式を指定するときに使用します。BLOB 式のデータ型は、LONG BINARY と見なされます。VALUE 句の値として、READ_CLIENT_FILE 関数を指定できます。

  FILE キーワードと VALUE キーワードをどちらも指定しないと、VALUE が指定されたと見なされます。

  FORMAT SHAPEFILE を使用した場合、FILE のみが指定されたと見なされます。

  WITH 句 この句は、問い合わせられるデータのローセットスキーマ (カラム名とデータ型) を指定するときに使用します。カラムは直接指定できます (たとえば、WITH (Surname CHAR(30), GivenName CHAR(30)))。また、TABLE サブ句を使用して、スキーマ情報の取得元に使用するテーブルを参照できます (たとえば、WITH TABLE dba.Employees (Surname, GivenName))。指定するテーブルの所有者であるか、または SELECT パーミッションが必要です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラムを指定する場合は、入力データ内でスキップするカラムに filler( ) を指定できます (たとえば、WITH (filler( ), Surname CHAR(30), GivenName CHAR(30)))。

OPTION 句　OPTION 句は、エスケープ文字、デリミター、エンコードなど、入力ファイルに使用する解析オプションを指定するときに使用します。サポートされるオプションは、入力ファイルの解析を制御する LOAD TABLE 文のオプションで構成されます。

- ***scan-option***　各スキャンオプションの詳細については、「LOAD TABLE 文」814 ページの load-option の説明を参照してください。

- ***apply-expression***　この句は、左の *table-expression* のローごとに、右の *table-expression* を評価するジョイン条件を指定するときに使用します。たとえば、適用式を使用して、テーブル式のローごとに関数、プロシージャー、または派生テーブルを評価できます。

- ***contains-expression***　テーブル名の後に付ける CONTAINS 句は、テーブルをフィルターして、*contains-query* で指定した全文クエリに一致するローのみを返すときに使用します。この句を指定すると、*score-correlation-name* を使用して参照できるスコアカラムとともに、テーブルの一致するローがすべて返されます。*score-correlation-name* を指定しない場合は、デフォルトの相関名 contains によってスコアカラムを参照できます。

  オプションの相関名引数を除き、CONTAINS 句は CONTAINS 探索条件の引数と同じ引数を取ります。

  CONTAINS 句にリストされるカラムには、テキストインデックスが必要です。

  contains-query は、NULL または空の文字列に設定できません。テキスト設定オブジェクトの設定により *contains-query* 内のすべての単語が削除されるようになっている場合、*contains-expression* によって参照されるベーステーブルのローは返されません。

- ***correlation-name***　*correlation-name* は、FROM 句の中でテーブルまたはビューに代替名を指定するときに使用します。この代替名は、文のどこからでも参照できます。たとえば、emp と dep はそれぞれ、Employees テーブルと Departments テーブルの相関名です。

```
SELECT Surname, GivenName, DepartmentName
  FROM Employees emp, Departments dep,
  WHERE emp.DepartmentID=dep.DepartmentID;
```

- ***dml-statement***　*dml-statement* を使用して、ローの選択に使用する DML 文 (INSERT、DELETE、UPDATE、または MERGE) を指定します。実行時には、*dml-derived-table* で指定された DML 文が最初に実行され、その DML の影響を受けるローが、REFERENCING 句で記述されているカラムを含むテンポラリテーブルにマテリアライズされます。テンポラリテーブルは、*dml-derived-table* の結果セットを表します。

  クエリ内で参照しないために結果をテンポラリテーブルにマテリアライズする必要がない場合は、REFERENCING ( ) または REFERENCING (NONE) を使用します。

  REFERENCING ( ) または REFERENCING (NONE) を指定した場合、更新されたローはクエリで参照されないため、*dml-derived-table* の結果セットを表すテンポラリテーブルにマテリアライズされません。この場合のテンポラリテーブルは空のテーブルになります。メインの文が実行される前に *dml-statement* が実行されるようにするには、この機能を使用します。

結果の OLD カラムには、更新操作の対象にするローを検索するスキャンで見つかった値が格納されます。FINAL カラムには、参照整合性チェックが実行され、計算カラムとデフォルトカラムが更新され、すべてのトリガー (FOR STATEMENT タイプの AFTER トリガーを除く) が起動された後に、値が格納されます。

文	サポートされているテーブルバージョン
INSERT	FINAL
DELETE	OLD
UPDATE	FINAL か OLD またはその両方
MERGE	FINAL か OLD またはその両方

OLD と FINAL の両方の名前を指定した場合、2 つの相関名が使用されます。ただし、これらの名前はどちらも同じ結果セットを示すため、本当の相関ではありません。REFERENCING (OLD AS O FINAL AS F) と指定した場合、暗黙的なジョイン述部 O.rowid = F.rowid が含まれます。 O.rowid = F.rowid.

INSERT 文では、FINAL のみがサポートされています。したがって、INSERT ON EXISTING UPDATE 文によって変更された更新ローの値は、派生テーブルの結果セットには含まれません。代わりに、MERGE 文を使用して insert-else-update 処理を実行します。

dml-derived-table 文は更新可能なテーブルを 1 つのみ参照できます。複数のテーブルに対する更新の場合は、エラーが返されます。また、DML 文が相関サブクエリまたは共通テーブル式に含まれている場合、これらの構成体のセマンティックが不明確になる可能性があるため、*dml-statement* からの選択は許可されません。

- **WITH *table-hint* 句**　WITH *table-hint* 句を使用すると、このテーブルでのみ使用できる動作と、この文でのみ使用できる動作を指定できます。この句を使用すると、独立性レベルを変更せずに、またデータベースまたは接続のオプションを設定せずに、動作を変更できます。テーブルヒントは、ベーステーブル、テンポラリテーブル、およびマテリアライズドビューに使用できます。

警告
WITH *table-hint* 句は高度な機能です。必要な場合にかぎり、経験を有するデータベース管理者のみが使用してください。また、この設定はすべての状況で適用されるとはかぎりません。

  - **独立性レベル関連のテーブルヒント**　独立性レベルのテーブルヒントは、テーブルのクエリを実行するときに独立性レベルの動作を指定する場合に使用します。また、指定したテーブルと現在のクエリにのみ使用するロック方法を指定します。スナップショットの独立性レベルをテーブルヒントとして指定することはできません。

    サポートされている独立性レベル関連のテーブルヒントの一覧を示します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

テーブルヒント	説明
HOLDLOCK	独立性レベル 3 と同等の動作を設定します。このテーブルヒントは SERIALIZABLE と同義です。
NOLOCK	独立性レベル 0 と同等の動作を設定します。このテーブルヒントは READUNCOMMITTED と同義です。
READCOMMITTED	独立性レベル 1 と同等な動作を設定します。
READPAST	書き込みロックがかけられたローをブロックするのではなく、無視するようにデータベースサーバーに指示します。このテーブルヒントは独立性レベル 1 でのみ使用できます。READPAST ヒントは、FROM 句の中の相関名がベーステーブルまたは共有されたグローバルテンポラリテーブルを参照する場合のみ考慮されます。それ以外の場合 (ビュー、プロキシテーブル、およびテーブル関数)、READPAST ヒントは無視されます。ベーステーブルの相関名にヒントを指定すると、ビュー内のクエリから READPAST を使用する場合があります。READPAST テーブルヒントを使用すると、サーバー内でロックと述部評価が相互に影響し合うため、異常事態が発生する可能性があります。また、DELETE 文、INSERT 文、または UPDATE 文のターゲットとなるテーブルに READPAST ヒントは使用できません。
READUNCOMMITTED	独立性レベル 0 と同等の動作を設定します。このテーブルヒントは NOLOCK と同義です。
REPEATABLEREAD	独立性レベル 2 と同等な動作を設定します。
SERIALIZABLE	独立性レベル 3 と同等の動作を設定します。このテーブルヒントは HOLDLOCK と同義です。

テーブルヒント	説明
UPDLOCK	ヒントが指定されたテーブルの文によって処理されるローを意図的ロックを使用してロックすることを指定します。影響を受けるローは、トランザクションの終わりまでロックされたままになります。UPDLOCK はすべての独立性レベルで機能し、意図的ロックを使用します。
XLOCK	ヒントが指定されたテーブルの文によって処理されるローを排他的にロックすることを指定します。影響を受けるローは、トランザクションの終わりまでロックされたままになります。XLOCK はすべての独立性レベルで機能し、書き込みロックを使用します。

注意
Mobile Link 同期に参加しているデータベースにクエリを書き込んでいる場合、同期スクリプトに READPAST テーブルヒントを使用しないことをおすすめします。

詳細については、次の項を参照してください。

- 「download_cursor テーブルイベント」『Mobile Link サーバー管理』
- 「download_delete_cursor テーブルイベント」『Mobile Link サーバー管理』
- 「upload_fetch テーブルイベント」『Mobile Link サーバー管理』

アプリケーションの更新回数が多すぎてダウンロードのパフォーマンスに影響が出ているために READPAST を考慮する場合、代替策としてスナップショットアイソレーションを使用する方法があります。

○ **最適化テーブルヒント (FASTFIRSTROW)**　FASTFIRSTROW テーブルヒントを使用すると、optimization_goal オプションを First-row に設定することなく、クエリの最適化ゴールを設定できます。FASTFIRSTROW を使用すると、SQL Anywhere は、クエリ結果の最初のローをフェッチする時間を短縮するためのアクセスプランを選択します。

● **WITH (*index-hint*) 句**　WITH (*index-hint*) 句を使用すると、クエリオプティマイザープラン選択アルゴリズムを無効にするインデックスヒントを指定できます。また、インデックスを使用してテーブルにアクセスする方法をオプティマイザーに正確に指示できます。インデックスヒントは、ベーステーブル、テンポラリテーブル、およびマテリアライズドビューに使用できます。

○ **NO INDEX**　この句は、強制的にテーブルを逐次スキャンするときに使用します (インデックスは不使用)。逐次スキャンのコストは非常に高くなる場合があります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **INDEX ([PRIMARY KEY | FOREIGN KEY] *index-name* [,...])**　この句は、オプティマイザーがクエリを処理するために使用する必要のあるインデックスを、最大 4 つまで指定するときに使用します。

指定したインデックスが 1 つでも使用できなかった場合はエラーが返されます。

PRIMARY KEY または FOREIGN KEY を指定して、テーブルの PRIMARY KEY インデックスと FOREIGN KEY インデックスの名前が同じ場合のあいまいさを排除できます。

PRIMARY キーまたは FOREIGN キーを指定しないでインデックスヒントにインデックス名を指定すると、テーブルに同じ名前を持つ複数のインデックスが存在する場合、オプティマイザーは標準のインデックスを選択します。標準のインデックスが存在しない場合、オプティマイザーはプライマリキーインデックスを選択します。プライマリキーインデックスが存在しない場合は、外部キーインデックスが代わりに使用されます。

index-name は、インデックスのユーザー ID とテーブル名を指定することで修飾できます。

INDEX 句で指定するインデックスは、そのテーブルに定義されているインデックスにする必要があります。それ以外の場合、エラーが返されます。たとえば、FROM Products WITH(INDEX (Products.xx)) は、インデックス xx が Products テーブルに定義されていない場合、エラーを返します。同様に、FROM Products WITH(INDEX (sales_order_items.sales_order_items)) は、sales_order_items.sales_order_items インデックスが存在していても、Products テーブルに定義されていないため、エラーを返します。

○ **INDEX ONLY { ON | OFF }**　この句は、データのインデックス専用取得を実行するかどうかを制御するときに使用します。INDEX ONLY ON を使用して INDEX (*index-name...*) 句を指定すると、データベースサーバーは、指定されたインデックスを使用してインデックス専用取得を実行しようとします。インデックス専用取得の処理で、指定したインデックスが 1 つでも使用できなかった場合は、エラーが返されます (たとえば、インデックスがない場合、または既存のインデックスがクエリを処理できない場合)。

INDEX ONLY OFF を指定すると、インデックス専用取得は行われません。

● **FORCE INDEX (*index-name*)**　FORCE INDEX (*index-name*) 構文は互換性のために用意されています。また、複数インデックスの指定はサポートされません。この句は、WITH (INDEX (index-name)) と同じです。

● **CLUSTERED INDEX**　この句は、オプティマイザーでクラスタードインデックスを使用する必要があること (存在する場合) を指定するときに使用します。ベーステーブル用に存在できるクラスタードインデックスは 1 つのみのため、インデックス名は指定しません。クラスタードインデックスが存在しないか使用できない場合は、エラーが返されます。

備考

FROM 句でストアドプロシージャーの引数としてサブクエリを指定することはできません。たとえば、次の文はエラーを返します。

```
SELECT *, ( SELECT 12 x ) D
FROM sa_rowgenerator( 1,( SELECT 12 x ) ):
```

SELECT 文、UPDATE 文、DELETE 文には、文が使用するテーブルを指定するテーブルリストが必要です。

> **注意**
> FROM 句の説明はテーブルについてのものですが、特に注意書きがなければビューと派生テーブルにも適用します。

FROM 句は、指定した全テーブルのすべてのカラムで構成される結果セットを作成します。最初に、コンポーネントテーブルのすべてのローの組み合わせが結果セットの中に入ります。次に、JOIN 条件か WHERE 条件、またはその両方の分だけ、通常は組み合わせの数が減ります。

CROSS JOIN には ON フレーズを使用できません。

パーミッション

openstring-expression の FILE 句には DBA 権限または READFILE 権限が必要です。

openstring-expression の TABLE 句では、指定されたテーブルをユーザーが所有しているか、または指定されたテーブルの SELECT パーミッションがユーザーに必要です。

関連する動作

なし

参照

- 「グループが所有するテーブルの参照」『SQL Anywhere サーバー データベース管理』
- 「CONTAINS 探索条件」51 ページ
- 「テキストインデックスの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「optimization_goal オプション」『SQL Anywhere サーバー データベース管理』
- 「書き込みロック」『SQL Anywhere サーバー SQL の使用法』
- 「意図的ロック」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの作成または変更時の指定項目」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの例」『SQL Anywhere サーバー SQL の使用法』
- 「DML 文に対する SELECT」『SQL Anywhere サーバー SQL の使用法』
- 「データ修正文」『SQL Anywhere サーバー SQL の使用法』
- 「独立性レベルと一貫性」『SQL Anywhere サーバー SQL の使用法』
- 「Mobile Link 独立性レベル」『Mobile Link サーバー管理』
- 「LOAD TABLE 文」814 ページ
- 「DELETE 文」697 ページ
- 「SELECT 文」895 ページ
- 「UPDATE 文」976 ページ
- 「INSERT 文」802 ページ
- 「MERGE 文」834 ページ
- 「派生テーブルのクエリ」『SQL Anywhere サーバー SQL の使用法』
- 「ジョイン：複数テーブルからのデータ検索」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

SQL/2008　FROM 句は、SQL/2008 標準の基礎部分です。FROM 句の複雑さは、FROM 句の個々のコンポーネントを標準の該当部分に照らしてチェックする必要があることを意味します。次に、SQL Anywhere でサポートされている、オプションの SQL/2008 言語機能の一部のリストを示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○CROSS JOIN、FULL OUTER JOIN、NATURAL JOIN は、オプションの SQL/2008 機能 F401 を構成します。

○INTERSECT と INTERSECT ALL は、オプションの SQL/2008 機能 F302 を構成します。

○EXCEPT ALL は、オプションの言語機能 F304 です。

○派生テーブルは、SQL/2008 言語機能 F591 です。

○FROM 句のプロシージャー (テーブル関数) は、機能 T326 です。SQL/2008 標準では TABLE キーワードを使用してプロシージャーの出力をテーブル式として識別する必要があるのに対し、SQL Anywhere では TABLE キーワードは不要です。

○共通テーブル式は、オプションの SQL/2008 言語機能 T121 です。共通テーブル式の中でネストされた派生テーブルで別の共通テーブル式を使用する方法は、言語機能 T122 です。

○再帰テーブル式は、機能 T131 です。共通テーブル式の中でネストされた派生テーブルでの、再帰テーブル式の使用は、オプションの SQL/2008 言語機能 T132 です。

次に示す FROM 句のコンポーネントはベンダー拡張です。

○KEY JOIN。

○CROSS APPLY と OUTER APPLY。

○OPENSTRING。

○CONTAINS を使用した *table-expression* (全文検索)。

○*dml-statement* を派生テーブルとして指定。

○すべてのテーブルヒント (WITH、FORCE INDEX、READPAST、独立性レベルのヒントの使用を含む)。

○ベンダー拡張の LATERAL (*table-expression*)。LATERAL (*select-statement*) は、SQL/2008 標準のオプションの SQL 言語機能 T491 です。

例

次は、有効な FROM 句です。

```
...
FROM Employees
...
```

```
...
FROM Employees NATURAL JOIN Departments
...
```

```
...
FROM Customers
KEY JOIN SalesOrders
KEY JOIN SalesOrderItems
KEY JOIN Products
...
```

```
...
FROM Employees CONTAINS ( Street, ' Way ' )
...
```

次のクエリは、クエリ内での派生テーブルの使い方を示します。

```
SELECT Surname, GivenName, number_of_orders
FROM Customers JOIN
    ( SELECT CustomerID, COUNT(*)
      FROM SalesOrders
       GROUP BY CustomerID )
    AS sales_order_counts( CustomerID,
                           number_of_orders )
ON ( Customers.ID = sales_order_counts.CustomerID )
WHERE number_of_orders > 3;
```

次のクエリは、ストアドプロシージャーの結果セットからローを選択する方法を示します。

```
SELECT t.ID, t.QuantityOrdered AS q, p.name
FROM ShowCustomerProducts( 149 ) t JOIN Products p
ON t.ID = p.ID;
```

次の例は、ファイルを問い合わせる OPENSTRING 句を使用して、クエリを実行する方法を示します。CREATE TABLE 文は、2 つのカラム column1 と columns2 を持つテーブル testtable を作成します。UNLOAD 文は、RowGenerator テーブルからローをアンロードして、ファイル *testfile.dat* を作成します。SELECT 文では FROM 句の中で OPENSTRING 句を指定し、testtable テーブルと RowGenerator テーブルの両方のスキーマ情報を使用して *testfile.dat* を問い合わせます。このクエリは値が 49 のローを 1 つ返します。

```
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;
```

次の例は、文字列値を問い合わせる OPENSTRING 句を使用して、クエリを実行する方法を示します。SELECT 文では FROM 句の中で OPENSTRING 句を使用し、WITH 句で与えられるスキーマ情報を使用して文字列値を問い合わせます。クエリは、3 つのローを持つ 2 つのカラムを返します。

```
SELECT *
  FROM OPENSTRING( VALUE '1,"First"$2,"Second"$3,"Third"')
  WITH (c1 INT, c2 VARCHAR(30))
  OPTION ( DELIMITED BY ',' ROW DELIMITED BY '$')
  AS VALS
```

次の例は、データ修正文によって修正されたローを選択するためのクエリを実行する方法を示します。この例では、青色の品目の在庫が半分以上減ると、警告が発行されます。

```
SELECT old_products.name, old_products.quantity, final_products.quantity
FROM
( UPDATE 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;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GET DATA 文 [ESQL]

カーソルの現在のローの 1 つのカラムに対する文字列またはバイナリデータを取得します。

構文

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*

パラメーター

COLUMN 句 *column-num* の値は 1 から始まり、どのカラムのデータがフェッチされるかを示します。そのカラムは、文字列型またはバイナリ型で指定します。

OFFSET 句 *start-offset* は、フィールド値の中でスキップされるバイト数を示します。通常、これは、以前にフェッチされたバイトの数です。この GET DATA 文に対してフェッチされるバイト数は、ターゲットのホスト変数の長さによって決定されます。

WITH TEXTPTR 句 WITH TEXTPTR 句がある場合、テキストポインターが SQLDA の第 2 ホスト変数または SQLDA の第 2 フィールドに取り出されます。このテキストポインターは、Transact-SQL READ TEXT と WRITE TEXT 文と一緒に使えます。テキストポインターは 16 ビットのバイナリ値で、次のように宣言できます。

```
DECL_BINARY( 16 ) textptr_var;
```

WITH TEXTPTR 句は、長いデータ型 (LONG BINARY、LONG VARCHAR、TEXT、IMAGE) でのみ使用できます。それ以外のデータ型で使用すると、エラー INVALID_TEXTPTR_VALUE が返されます。

データの合計の長さが、SQLCA 構造体の SQLCOUNT フィールドに返されます。

USING DESCRIPTOR 句 *sqlda-name* では、フェッチされたデータを受信する SQLDA (SQL Descriptor Area) を指定します。USING DESCRIPTOR 句を使用すると、フェッチされたデータを受信するためのホスト変数を動的に指定できます。

INTO 句 INTO 句は、フェッチされたデータを受信するホスト変数を指定するときに使用します。ターゲットのホスト変数のインジケーター値は a_sql_len 型であり、現在は 16 ビット値です。このため、必ずしもトランケートされたバイト数が入るほど大きいわけではありません。代わりに、フィールドに NULL 値がある場合は負の値が入り、値がトランケートされた場合は (トランケートされたバイトの数とはかぎりません) 正の値が入ります。また、NULL 値以外の値がトランケートされない場合は、0 が入ります。

また、LONG VARCHAR、LONG NVARCHAR、または LONG BINARY ホスト変数で 0 より大きいオフセットを使用すると、untrunc_len フィールドにはトランケート前のサイズが正確に示されません。

備考

現在のカーソルの位置でローから 1 つのカラム値を取得します。カーソルを開き、FETCH を使ってローに配置しておいてください。

GET DATA を使って、LONG BINARY または LONG VARCHAR フィールドをフェッチします。

パーミッション

なし

関連する動作

なし

参照

- 「FETCH 文 [ESQL] [SP]」749 ページ
- 「READTEXT 文 [T-SQL]」866 ページ
- 「SET 文」927 ページ
- INTO 句 771 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、GET DATA を使用してバイナリの大規模なオブジェクト (BLOB とも呼ばれます) をフェッチします。

```
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;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GET DESCRIPTOR 文 [ESQL]

記述子領域内の変数に関する情報を取り出すか、その値を取り出します。

構文

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*

備考

GET DESCRIPTOR 文は、記述子領域内の変数に関する情報を取り出すか、その値を取り出すために使用します。

値 { *integer* | *hostvar* } には、情報を取り出す記述子領域内の変数を指定します。GET...DATA の実行中に型チェックが実行され、ホスト変数と記述子変数のデータ型が同じかどうか確認されます。LONG VARCHAR と LONG BINARY は GET DESCRIPTOR...DATA ではサポートされていません。

エラーが発生すると、戻り値は SQLCA に格納されます。

パーミッション

なし

関連する動作

なし

参照

- 「ALLOCATE DESCRIPTOR 文 [ESQL]」406 ページ
- 「DEALLOCATE DESCRIPTOR 文 [ESQL]」684 ページ
- 「SET DESCRIPTOR 文 [ESQL]」907 ページ
- 「SQLDA (SQL descriptor area)」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　GET DESCRIPTOR は、オプションの SQL/2008 言語機能 B031、"Basic dynamic SQL" の一部です。

例

次の例は、sqlda の位置 col_num にあるカラムの型を返します。

```
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 );
}
```

GET OPTION 文 [ESQL]

オプションの現在の設定を取得します。この文の代わりに、CONNECTION_PROPERTY 関数を使用することをおすすめします。

構文

GET OPTION [*userid*.]*option-name*
{ **INTO** *hostvar* | **USING DESCRIPTOR** *sqlda-name* }

userid : *identifier*, *string* | *hostvar*

option-name :
identifier
| *string*
| *hostvar*

hostvar : 許可されたインジケーター変数

sqlda-name : *identifier*

備考

GET OPTION 文は、このソフトウェアの旧バージョンとの互換性のために用意されています。オプションの値を取得するには、CONNECTION_PROPERTY システム関数の使用をおすすめします。

GET OPTION 文は、*option-name* のオプション設定を取得します。このオプション設定は *userid* に対するものか、または *userid* を指定していない場合は接続しているユーザーに対するものです。これはユーザー個人の設定であるか、または、接続しているユーザーに設定がない場合は PUBLIC 設定となります。指定したオプションがデータベースオプションであり、ユーザーがそのオプションに対してテンポラリ設定を持っている場合、テンポラリ設定が取得されます。

option-name が存在しない場合、GET OPTION 文は警告 SQLE_NOTFOUND を返します。

パーミッション

何も必要ありません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

なし

参照

- 「SET OPTION 文」913 ページ
- 「システムプロシージャーのアルファベット順リスト」1035 ページ
- 「CONNECTION_PROPERTY 関数 [システム]」178 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、GET OPTION の使い方を示します。

```
EXEC SQL GET OPTION 'date_format' INTO :datefmt;
```

GOTO 文 [T-SQL]

ラベルの付いた文に制御を分岐します。

構文

label : **GOTO** *label*

備考

Transact-SQL プロシージャー、トリガー、バッチに含まれる任意の文には、ラベルを付けることができます。ラベル名は末尾にコロン (:) を付けた有効な識別子です。GOTO 文では、コロンは使用できません。

複合文をネストする場合、現在の複合文およびその上位の複合文内のラベルに進む (GOTO) だけです。上位の複合文内でネストされている他の複合文にあるラベルに進むことはできません。

パーミッション

なし

関連する動作

なし

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の Transact-SQL バッチは、データベースサーバーメッセージウィンドウに 4 回 "yes" のメッセージを表示します。

```
DECLARE @count SMALLINT
SELECT @count = 1
```

```
restart:
  PRINT 'yes'
  SELECT @count = @count + 1
  WHILE @count <=4
   GOTO restart
```

GRANT CONSOLIDATE 文 [SQL Remote]

SQL Remote 階層において、現在のデータベースのすぐ上にあり、現在のデータベースからメッセージを受信するデータベースを識別します。

構文

GRANT CONSOLIDATE
TO *userid*
TYPE *message-system*, ...
ADDRESS *address-string*, ...
[**SEND** { **EVERY** | **AT** } *hh*:*mm*:*ss*]

message-system :
FILE | **FTP** | **SMTP**

address : *string*

パラメーター

userid パーミッションが付与されるユーザーのユーザー ID。

message-system SQL Remote がサポートするメッセージシステムの 1 つ。

address 指定したメッセージシステムのアドレス。

備考

SQL Remote インストール環境で、SQL Remote 階層で現在のデータベースのすぐ上にあるデータベースに、CONSOLIDATE パーミッションを付与します。GRANT CONSOLIDATE は、統合データベースを識別するためにリモートデータベースで発行されます。各データベースには、CONSOLIDATE パーミッションを持つユーザー ID を 1 つしか指定できません。つまり、1 つのデータベースを複数の統合データベースのリモートデータベースにすることはできません。

統合ユーザーはメッセージシステムにより識別され、統合ユーザーに対するメッセージの送受信方法が識別されます。address-name には、メッセージシステムに対して有効なアドレスを、一重引用符で囲んで指定します。統合ユーザーは、各リモートデータベースに対して 1 人だけです。

FILE メッセージタイプの場合、アドレスは SQLREMOTE 環境変数で指定されているディレクトリのサブフォルダーです。

GRANT CONSOLIDATE 文は、統合データベースがメッセージを受信するためには必要ですが、統合データベースをサブスクライブして何らかのデータを取得するものではありません。データをサブスクライブするには、現在のデータベースにあるパブリケーションの統合ユーザー ID に対するサブスクリプションを作成します。統合データベースでデータベース抽出ユーティリティを実行すると、それに対して適切な GRANT CONSOLIDATE 文が実行された状態でリモートデータベースが作成されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプションの SEND EVERY 句と SEND AT 句を使用すると、メッセージを送信する間隔を指定できます。文字列には、メッセージ送信間隔の時間 (SEND EVERY) またはメッセージを送信する時刻 (SEND AT) を指定します。SEND AT の場合、メッセージは 1 日に 1 回送信されます。

ユーザーに SEND EVERY 句も SEND AT 句もない REMOTE パーミッションが付与されている場合、Message Agent はメッセージを処理した後、停止します。Message Agent を継続的に実行する場合は、REMOTE パーミッションを持つすべてのユーザーの SEND AT または SEND EVERY の頻度が指定されていることを確認します。

多くのリモートデータベースでは Message Agent を定期的に実行するため、統合データベースに SEND 句が指定されていないことがあります。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「CONSOLIDATE パーミッション」『SQL Remote』
- 「GRANT PUBLISH 文 [SQL Remote]」777 ページ
- 「GRANT REMOTE 文 [SQL Remote]」780 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」884 ページ
- 「ALTER REMOTE MESSAGE TYPE 文 [SQL Remote]」436 ページ
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」613 ページ
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」645 ページ
- 「CONSOLIDATE パーミッションの付与」『SQL Remote』
- 「SQL Remote メッセージシステム」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

```
GRANT CONSOLIDATE TO con_db
TYPE SMTP
ADDRESS 'Singer, Samuel';
```

GRANT PUBLISH 文 [SQL Remote]

現在のデータベースのパブリッシャーを識別します。

構文

GRANT PUBLISH TO *userid*

備考

SQL Remote インストール環境内の各データベースは、出力メッセージ内でユーザー ID により識別され、「パブリッシャー」と呼ばれます。GRANT PUBLISH 文は、これらの出力メッセージと関連付けられているパブリッシャーユーザー ID を識別します。

PUBLISH 権限は 1 つのユーザー ID にしか割り当てられません。PUBLISH 権限を持つユーザー ID は、特別値 CURRENT PUBLISHER で識別します。次のクエリは、現在のパブリッシャーを識別します。

```
SELECT CURRENT PUBLISHER;
```

パブリッシャーがない場合、この特殊定数の値は NULL になります。

CURRENT PUBLISHER 特別値を、カラムのデフォルトの設定として使用できます。CURRENT PUBLISHER カラムをテーブルのレプリケーションを行う際のプライマリキーに含めると、複数サイトでのアップデートによりプライマリキーの矛盾が発生するのを回避できるので便利です。

パブリッシャーを変更するには、まず REVOKE PUBLISH 文を使って現在のパブリッシャーを削除し、それから GRANT PUBLISH 文を使って新しいパブリッシャーを作成します。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「PUBLISH パーミッション」『SQL Remote』
- 「GRANT CONSOLIDATE 文 [SQL Remote]」776 ページ
- 「REVOKE PUBLISH 文 [SQL Remote]」885 ページ
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」645 ページ
- 「CURRENT PUBLISHER 特別値」62 ページ
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」613 ページ
- 「GRANT REMOTE 文 [SQL Remote]」780 ページ
- 「パブリッシャーの作成」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

```
GRANT PUBLISH TO publisher_ID;
```

GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]

ユーザー ID にリモート DBA 権限を付与します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

GRANT REMOTE DBA
TO *userid*, ...
[**IDENTIFIED BY** *password*]

パラメーター

IDENTIFIED BY 句　IDENTIFIED BY 句は、この文のオプションです。含めると、ユーザーのパスワードが変更されます。

備考

REMOTE DBA 権限を持つユーザー ID が完全な DBA 権限を持つのは、次の場合のみです。

- Mobile Link において、接続が SQL Anywhere 同期クライアント (dbmlsync) ユーティリティから行われた場合。REMOTE DBA 権限で、dbmlsync にデータベースへのフルアクセス権を付与することができます。同じユーザー ID を使用する他の接続には、特別な権限は付与されません。

- SQL Remote において、接続が Message Agent から行われた場合。REMOTE DBA 権限があると、Message Agent はデータベースに対するフルアクセス権を持ち、メッセージに含まれている任意の変更を行うことができます。同じユーザー ID を使用する他の接続には、特別な権限は付与されません。

REMOTE DBA 権限は、ユーザー ID に完全な DBA 権限を付与しなくて済むようにして、DBA のユーザー ID やパスワードの配布に関連するセキュリティ問題を回避します。

たとえば、REMOTE DBA 権限を持つ SQL Remote ユーザー ID は、Message Agent 以外からの接続に対して特別な権限を持ちません。たとえ REMOTE DBA ユーザーのユーザー ID とパスワードが公開されていても、セキュリティ上の問題は発生しません。そのデータベースで CONNECT より上のパーミッションが付与されたユーザー ID を使用しないかぎり、データベース内のデータにアクセスすることはできません。

パーミッション

DBA 権限が必要です。

関連する動作

オートコミット。

参照

- Mobile Link：「同期の開始」『Mobile Link クライアント管理』
- SQL Remote：「REMOTE DBA 権限」『SQL Remote』
- SQL Remote：「REMOTE DBA 権限の付与」『SQL Remote』
- SQL Remote：「REMOTE DBA 権限の取り消し」『SQL Remote』
- SQL Remote：「SQL Remote Message Agent (dbremote) モード」『SQL Remote』
- SQL Remote：「セキュリティ」『SQL Remote』
- 「REMOTE DBA 権限」『SQL Anywhere サーバー データベース管理』
- 「REVOKE REMOTE DBA 文 [SQL Remote]」886 ページ
- 「dbmlsync のパーミッション」『Mobile Link クライアント管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

REMOTE DBA 権限を **dbremote** というユーザー ID に付与するには、次のように指定します。

```
GRANT REMOTE DBA
TO dbremote
IDENTIFIED BY dbremote;
```

GRANT REMOTE 文 [SQL Remote]

SQL Remote 階層において、現在のデータベースのすぐ下にあり、現在のデータベースからメッセージを受信するデータベースを識別します。これらはリモートユーザーといいます。

構文

GRANT REMOTE TO *userid*, ...
TYPE *message-system*, ...
ADDRESS *address-string*, ...
[**SEND** { **EVERY** | **AT** } *send-time*]

パラメーター

userid　パーミッションを付与されるユーザーのユーザー ID。

message-system　SQL Remote がサポートするメッセージシステムの 1 つ。次のいずれかの値を指定します。

- FILE
- FTP
- SMTP

address-string　指定したメッセージシステムに対して有効なアドレスを含む文字列。

send-time　*hh*:*mm*:*ss* の形式で時刻を指定した文字列。

備考

SQL Remote インストール環境では、現在のデータベースからメッセージを受信する各データベースに REMOTE パーミッションが必要です。

この唯一の例外として、SQL Remote 階層で現在のデータベースのすぐ上にあるデータベースに、CONSOLIDATE パーミッションを付与します。

リモートユーザーはメッセージシステムにより識別され、統合ユーザーに対するメッセージの送受信方法が識別されます。address-name には、メッセージシステムに対して有効なアドレスを、一重引用符で囲んで指定します。

FILE メッセージタイプの場合、アドレスは SQLREMOTE 環境変数で指定されているディレクトリのサブフォルダーです。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GRANT REMOTE 文は、リモートデータベースがメッセージを受信するのには必要ですが、リモートデータベースをサブスクライブして何らかのデータを取得するものではありません。データをサブスクライブするには、データベース抽出ユーティリティまたは CREATE SUBSCRIPTION 文を使用して、現在のデータベースにあるパブリケーションのユーザー ID に対するサブスクリプションを作成します。

オプションの SEND EVERY 句と SEND AT 句を使用すると、メッセージを送信する間隔を指定できます。文字列には、メッセージ送信間隔の時間 (SEND EVERY) またはメッセージを送信する時刻 (SEND AT) を指定します。SEND AT の場合、メッセージは 1 日に 1 回送信されます。

ユーザーに SEND EVERY 句も SEND AT 句もない REMOTE パーミッションが付与されている場合、Message Agent はメッセージを処理した後、停止します。Message Agent を継続的に実行する場合は、REMOTE パーミッションを持つすべてのユーザーの SEND AT または SEND EVERY の頻度が指定されていることを確認します。

多くの統合データベースでは Message Agent を定期的に実行するため、すべてのリモートデータベースに SEND 句が指定されている必要があります。ラップトップユーザーに毎日 (SEND AT)、リモートサーバーに 1、2 時間ごとに (SEND EVERY) メッセージを送信するというのが、一般的な設定です。効率的に運用するためには、この設定の時間をできるだけ変えずに使用してください。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「REMOTE パーミッション」『SQL Remote』
- 「REMOTE パーミッションの付与」『SQL Remote』
- 「サブスクリプション」『SQL Remote』
- 「送信頻度の設定」『SQL Remote』
- 「SQL Remote メッセージシステム」『SQL Remote』
- 「GRANT PUBLISH 文 [SQL Remote]」777 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」887 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」776 ページ
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」645 ページ
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」613 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、REMOTE パーミッションをユーザー SamS に付与し、SMTP 電子メールシステムを使用して、アドレス Singer, Samuel に 2 時間ごとにメッセージを送信するように設定します。

```
GRANT REMOTE TO SamS
TYPE SMTP
```

```
ADDRESS 'Singer, Samuel'
SEND EVERY '02:00';
```

GRANT 文

グループのメンバーシップの付与、新しいユーザー ID の作成、指定したユーザーに対するパーミッションの付与またはそのパーミッションの変更、パスワードの作成または変更を行います。

構文 1 - 権限の付与

GRANT *authority*, ...
TO *userid*, ...

authority :
BACKUP
| **DBA**
| **PROFILE**
| **READCLIENTFILE**
| **READFILE**
| [**RESOURCE** | **ALL**]
| **VALIDATE**
| **WRITECLIENTFILE**

構文 2 - グループステータスまたはグループのメンバーシップの付与

GRANT { **GROUP** | **MEMBERSHIP IN GROUP** *userid*, ... }
TO *userid*, ...

構文 3 - テーブルとビューパーミッションの付与

GRANT *permission*, ...
ON [*owner*.]*object-name*
TO *userid*, ...
[**WITH GRANT OPTION**]
[**FROM** *userid*]

permission :
ALL [**PRIVILEGES**]
| **ALTER**
| **DELETE**
| **INSERT**
| **REFERENCES** [(*column-name*, ...)]
| **SELECT** [(*column-name*, ...)]
| **UPDATE** [(*column-name*, ...)]

構文 4 - 実行パーミッションの付与

GRANT EXECUTE ON [*owner*.]{ *procedure-name* | *user-defined-function* }
TO *userid*, ...

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 5 - 統合化ログインの付与

GRANT INTEGRATED LOGIN TO *user-profile-name*, ...
AS USER *userid*

構文 6 - Kerberos ログインの付与

GRANT KERBEROS LOGIN TO *client-Kerberos-principal*, ...
AS USER *userid*

構文 7 - 接続パーミッションの付与

GRANT CONNECT TO *userid*, ...
[**AT** *starting-id*]
[**IDENTIFIED BY** *password*, ...]

構文 8 - DB 領域への作成パーミッションの付与

GRANT CREATE ON *dbspace-name*
TO *userid*, ...

構文 9 - シーケンスに対するパーミッションの付与

GRANT USAGE ON SEQUENCE *sequence-name*
TO *userid*, ...

パラメーター

AT *starting-id* 句　この句は一般的には使用されません。この句では、リストの最初のユーザー ID に使用する内部数値を指定します。

AT starting-id 句は、アンロードユーティリティが使用します。

GRANT authority 句　この句は、次に示す権限のいずれかを付与するときに使用します。

- **BACKUP 権限**　この権限を付与すると、ユーザーはデータベースをバックアップできるようになります。
- **DBA 権限**　この権限を付与すると、ユーザーはすべてのタスクを実行できるようになります。通常、これはデータベース管理者のために予約されています。
- **PROFILE 権限**　この権限を付与すると、ユーザーはプロファイリングと診断操作を実行できるようになります。
- **READCLIENTFILE 権限**　この権限を付与すると、データをロードするときなどに、ユーザーがクライアントコンピューター上のファイルを読み込むことができるようになります。
- **READFILE 権限**　この権限を付与すると、ユーザーは OPENSTRING 句を使用してファイルに SELECT 文を実行できるようになります。

○ **RESOURSE 権限または ALL 権限**　この権限を付与すると、ユーザーはテーブルとビューを作成できるようになります。ALL は Adaptive Server Enterprise と互換性のある RESOURCE の同意語です。

○ **VALIDATE 権限**　この権限を付与すると、ユーザーはさまざまな VALIDATE 文でサポートされる検証操作を実行できるようになります。この検証とは、たとえば、データベースの検証、テーブルとインデックスの検証、およびチェックサムの検証などです。また、ユーザーは Sybase Central の検証ユーティリティ (dbvalid) とデータベース検証ウィザードを使用できるようになります。

○ **WRITECLIENTFILE 権限**　この権限を付与すると、データをアンロードするときなどに、ユーザーがクライアントコンピューター上のファイルに書き込むことができるようになります。

GROUP 句　このパーミッションを使用すると、ユーザーがメンバーを持てるようになります。

MEMBERSHIP IN GROUP 句　このパーミッションは、グループのメンバーシップをユーザーに付与します。ユーザーは、グループレベルで設定された継承可能なパーミッションと権限を継承します。

GRANT *permission* 句　GRANT *permission* 句を使用すると、個々のテーブルまたはビューにパーミッションを付与できます。テーブルおよびビューパーミッションは、個々に指定したり、ALL を使用して一度にすべてのパーミッションを付与したりできます。次に、付与できるパーミッションのリストを示します。

○ **ALL パーミッション**　このパーミッションは、ALTER、DELETE、INSERT、REFERENCES、SELECT、および UPDATE の各パーミッションをテーブルに付与します。このパーミッションは、DELETE、INSERT、および UPDATE の各パーミッションをビューに付与します。

○ **ALTER パーミッション**　このパーミッションを使用すると、ユーザーが ALTER TABLE 文を使用して指定のテーブルを変更できます。このパーミッションは、ビューには使えません。

○ **DELETE パーミッション**　このパーミッションを使用すると、ユーザーは指定したテーブルまたはビューからローを削除できます。

○ **INSERT パーミッション**　このパーミッションを使用すると、ユーザーは指定したテーブルまたはビューにローを挿入できます。

○ **REFERENCES パーミッション**　このパーミッションを使用すると、ユーザーは指定したテーブルのインデックスを作成できます。また、指定したテーブルを参照する外部キーを作成できます。カラム名を指定する場合、ユーザーはそれらのカラムのみを参照できます。カラムの REFERENCES パーミッションは、ビューに対しては付与されません。テーブルだけに対して付与されます。INDEX は REFERENCES の同意語です。

○ **SELECT パーミッション**　このパーミッションを使用すると、ユーザーはビューまたはテーブル内の情報を参照できます。カラム名を指定する場合、ユーザーはそれらのカラムのみを参照できます。カラムの SELECT パーミッションは、ビューに対しては付与されません。テーブルだけに対して付与されます。

○ **UPDATE パーミッション**　このパーミッションを使用すると、ユーザーはビューまたはテーブル内のローを更新できます。カラム名を指定する場合、ユーザーはそれらのカラムのみを更新できます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

WITH GRANT OPTION 句　WITH GRANT OPTION を指定する場合、指定した名前のユーザー ID にもパーミッションが与えられ、他のユーザー ID に同じパーミッションを GRANT (付与) できます。グループのメンバーは、グループに付与されている WITH GRANT OPTION を継承しません。

FROM 句　FROM *userid* を指定すると、その *userid* は付与者のユーザー ID としてシステムテーブルに記録されます。この句は、アンロードユーティリティ (dbunload) での使用を目的としています。このオプションを直接使用したり変更したりしないでください。

CONNECT TO 句

> **注意**
> ユーザーを作成する場合は、CREATE USER 文を使用することをおすすめします。

新しいユーザーを作成します。GRANT CONNECT を使うと、どのユーザーも自分のパスワードを変更できます。パスワードとして空の文字列を持つユーザーを作成するには、次のように入力します。

```
GRANT CONNECT TO userid IDENTIFIED BY "";
```

パスワードのないユーザーを作成するには、次のように入力します。

```
GRANT CONNECT TO userid;
```

パスワードのないユーザーは、データベースに接続できません。これは、グループを作成し、他のユーザーはグループユーザー ID を使用してデータベースに接続させないようにする場合に便利です。ユーザー ID には、有効な ID を使用します。

○ユーザー ID には、次に該当する値を指定できません。

- 空白スペース、一重引用符、または二重引用符で始まる値
- 空白スペースで終わる値
- セミコロンを含む値

○パスワードは大文字と小文字が区別され、次に該当する値は指定できません。

- 空白スペース、一重引用符、または二重引用符で始まる値
- 空白スペースで終わる値
- セミコロンを含む値
- 256 バイト長を超える値

verify_password_function オプションは、パスワード規則 (1 桁以上の長さであることなど) の実装に使用できる関数を指定します。パスワード検証機能を使用する場合、GRANT CONNECT 文に複数のユーザー ID とパスワードを指定することはできません。

CREATE ON 句　ユーザーは、指定された DB 領域にデータベースオブジェクトを作成できます。CREATE パーミッションは、グループメンバーシップを通して継承できます。ユーザーがオブジェクトを作成するには、RESOURCE 権限も必要です。

GRANT USAGE ON SEQUENCE 句　ユーザーがシーケンス内の現在の値または次の値を評価できるようにします。この文を実行するには、DBA 権限が必要です。または、シーケンスの作

成者である必要があります。シーケンスがテーブルの DEFAULT 句の一部である場合、テーブルにローを挿入するすべてのユーザーにシーケンスに対するパーミッションが必要です。

備考

GRANT 文を使用して、データベースパーミッションと権限を個々のユーザー ID とグループに付与します。ユーザーとグループを作成するときにも使用します。

パーミッションは無効化されたオブジェクトに対して付与できます。無効化されたオブジェクトに対するパーミッションはデータベースに保存され、オブジェクトが有効になると使用できるようになります。

GRANT 文の構文 4 を使って、プロシージャーを実行するパーミッションを付与します。

GRANT 文の構文 5 は、1 つまたは複数の Windows ユーザーまたはグループプロファイルと既存のデータベースユーザー ID の間に明示的な統合化ログインマッピングを作成します。これにより、ローカルコンピューターにログインできたユーザーは、ユーザー ID またはパスワードを指定せずにデータベースに接続できます。*user-profile-name* は、*domain*¥*user-name* の形式にできます。統合化ログインがマッピングされているデータベースのユーザー ID にはパスワードを指定する必要があります。

GRANT 文の構文 6 は、1 つ以上の Kerberos プリンシパルから既存のデータベースユーザー ID にマッピングして、Kerberos の認証済みログインを作成します。その結果、Kerberos に正常にログインしたユーザー (有効な Kerberos のチケットが付与されているチケットを持つユーザー) は、ユーザー ID やパスワードを入力しなくてもデータベースに接続できます。Kerberos ログインがマッピングされているデータベースのユーザー ID にはパスワードを指定する必要があります。*client-Kerberos-principal* のフォーマットは、*user*/*instance*@*REALM* にします。この /*instance* はオプションです。領域を含む完全なプリンシパルと、インスタンスまたは領域を区別して扱う場合にのみ異なるプリンシパルを指定する必要があります。

プリンシパルの大文字と小文字は区別されるため、正しく指定する必要があります。大文字と小文字の違いしかない複数のプリンシパルのマッピングはサポートされていません (たとえば、jjordan@MYREALM.COM と JJordan@MYREALM.COM の両方にマッピングすることはできません)。

Kerberos プリンシパルに明示的なマッピングがなく、Guest データベースのユーザー ID が存在し、パスワードがある場合、Kerberos プリンシパルは Guest データベースのユーザー ID (統合化ログイン用と同じ Guest データベースのユーザー ID) を使用して接続します。

パーミッション

- **構文 3**　FROM 句を指定する場合、DBA 権限が必要です。それ以外の場合、テーブルの所有者であるか、テーブル WITH GRANT OPTION にパーミッションが付与されている必要があります。

- **構文 4**　プロシージャーの所有者であるか、DBA 権限が必要です。

- **構文 5 と 6**　DBA 権限が必要です。

- **構文 7**　自分のパスワードを変更するには、GRANT CONNECT を使用するか、DBA 権限が必要です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

オートコミット。

参照

- 「バックアップ権限」『SQL Anywhere サーバー データベース管理』
- 「CREATE USER 文」677 ページ
- 「リソース権限」『SQL Anywhere サーバー データベース管理』
- 「verify_password_function オプション」『SQL Anywhere サーバー データベース管理』
- 「グループ」『SQL Anywhere サーバー データベース管理』
- 「Kerberos 認証」『SQL Anywhere サーバー データベース管理』
- 「グループ」『SQL Anywhere サーバー データベース管理』
- 「WRITECLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』
- 「VALIDATE 権限」『SQL Anywhere サーバー データベース管理』
- 「リソース権限」『SQL Anywhere サーバー データベース管理』
- 「READFILE 権限」『SQL Anywhere サーバー データベース管理』
- 「プロファイル権限」『SQL Anywhere サーバー データベース管理』
- 「READCLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』
- 「DBA 権限」『SQL Anywhere サーバー データベース管理』
- 「ユニークな値を生成するためのシーケンスの使用」『SQL Anywhere サーバー SQL の使用法』
- 「Windows 統合ログイン」『SQL Anywhere サーバー データベース管理』
- 「REVOKE 文」887 ページ
- 「パーミッション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　構文 3 は、SQL/2008 標準のコア機能です。構文 3 では、ALTER 権限と同様に、FROM 句はベンダー拡張です。SQL/2008 標準では、任意の ALL PRIVILEGES 構文ではなく、PRIVILEGES キーワードが必須となります。

  構文 4 も SQL/2008 のコア機能であり、ストアドプロシージャーに対する EXECUTE パーミッションを付与するために使用されます。

  構文 9 は、オプションの SQL/2008 言語機能 T176 の一部です。

  他の構文はすべてベンダー拡張です。

例

新しいデータベースユーザーを作成します。

```
GRANT CONNECT TO SQLTester
IDENTIFIED BY welcome
```

Laurel というユーザーに Employees テーブル上のパーミッションを付与します。

```
GRANT
SELECT, UPDATE ( Street )
ON Employees
TO Laurel;
```

1 つの文で複数のパーミッションを付与できます。パーミッションはカンマで区切ります。

ユーザー Hardy に Calculate_Report プロシージャーの実行を許可します。

```
GRANT EXECUTE ON Calculate_Report
TO Hardy;
```

GROUP BY 句

カラム、エイリアス名、関数を SELECT 文の一部としてグループ化します。

構文

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* **)**
| **( )**

パラメーター

GROUPING SETS 句 GROUPING SETS 句を使用すると、1 つのクエリ指定から複数のグループ化に対して集約操作を実行できます。GROUPING SET 句で指定した各セットは、GROUP BY 句と同等です。

たとえば、次の 2 つのクエリは同じです。

```
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;
```

結果のローが所属するグループに応じて、グループ化の式を NULL 値として結果セットに反映することができます。この場合、NULL が別のグループ化の結果か、NULL が基本となるデータの実際の NULL 値の結果かについて混乱する可能性があります。入力データに存在する NULL 値とグループ化演算子によって挿入された NULL 値を区別するには、GROUPING 関数を使用します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GROUPING SETS 句に空のかっこ ( ) を指定すると、全体の集約を含む 1 つのローを返します。

ROLLUP 句　1 つのクエリ指定内に複数のグループ化を指定するときに使用できるという点で、ROLLUP 句は GROUPING SETS 句と似ています。*n simple-group-by-term* の ROLLUP 句は、*n*+1 のグループ化セットを生成します。フォーマットは空のかっこで始まり、左から右に連続する *group-by-term* を付加して構成されます。

たとえば、次の 2 つの文は同じです。

```
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, ( ) );
```

GROUPING SETS 句内で ROLLUP 句を使用できます。

CUBE 句　1 つのクエリ指定内に複数のグループ化を指定するときに使用できるという点で、CUBE 句は ROLLUP 句と GROUPING SETS 句に似ています。CUBE 句は、CUBE 句に表示されている式から作成されるすべての可能な組み合わせを表すときに使用します。

たとえば、次の 2 つの文は同じです。

```
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, () );
```

GROUPING SETS 句内で CUBE 句を使用できます。

WITH ROLLUP 句　これは ROLLUP 句の代わりになる構文であり、Transact-SQL との互換性のために提供されています。

WITH CUBE 句　これは CUBE 句の代わりになる構文であり、Transact-SQL との互換性のために提供されています。

備考

GROUP BY 句を使用している場合、式 (いくつかの制限付き)、カラム、エイリアス名、関数でグループ化できます。クエリの結果には、各グループ化セットの個々の値 (または複数の値) に 1 つのローが含まれます。

空の GROUP BY リスト () は、入力全体の処理を 1 つのグループとして表します。たとえば、次の 2 つの文は同じです。

```
SELECT COUNT(), SUM(Salary) FROM Employees;

SELECT COUNT(), SUM(Salary) FROM Employees GROUP BY ();
```

参照

- 「クエリ結果の要約、グループ化、ソート」『SQL Anywhere サーバー SQL の使用法』
- 「SELECT 文」895 ページ
- 「GROUP BY 句の拡張」『SQL Anywhere サーバー SQL の使用法』
- 「GROUPING 関数 [集合]」238 ページ
- 「GROUP BY と SQL/2008 標準」『SQL Anywhere サーバー SQL の使用法』
- 「空のグループ化指定の使用」『SQL Anywhere サーバー SQL の使用法』
- 「CUBE 句」『SQL Anywhere サーバー SQL の使用法』
- 「ROLLUP 句」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** GROUP BY は、SQL/2008 標準のコア機能です。GROUPING SETS、GROUP BY ()、ROLLUP、CUBE は、オプションの SQL/2008 言語機能 T431 の一部を構成します。SQL Anywhere では、オプションの SQL/2008 言語機能 T432、"Nested and concatenated GROUPING SETS" はサポートされていません。

  GROUP BY 句のベンダー拡張を次に示します。

  ○WITH ROLLUP

  ○WITH CUBE

  ○任意の式を GROUP BY の単語として指定する機能。SQL/2008 標準では、GROUP BY のすべての単語が、クエリの FROM 句内の基本となるテーブルからのカラム参照である必要があります。

例

次の例は、注文の総数を示す結果セットを返し、各年 (2000 と 2001) の注文数の小計を提供します。

```
SELECT year ( OrderDate ) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM SalesOrders
GROUP BY ROLLUP ( Year, Quarter )
ORDER BY Year, Quarter;
```

前の ROLLUP 演算の例と同様に、次の CUBE クエリの例は注文の総数を示す結果セットを返し、各年 (2000 と 2001) の注文数の小計を提供します。ROLLUP とは異なり、このクエリは各四半期 (1、2、3、4) の注文数の小計も示します。

```
SELECT year (OrderDate) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM SalesOrders
GROUP BY CUBE ( Year, Quarter )
ORDER BY Year, Quarter;
```

次の例は、2000 年と 2001 年の注文数の小計を示す結果セットを返します。GROUPING SETS 演算では、CUBE 演算のように小計の組み合わせをすべて返すのではなく、小計するカラムを選択します。

```
SELECT year (OrderDate) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM SalesOrders
GROUP BY GROUPING SETS ( ( Year, Quarter ), ( Year ) )
ORDER BY Year, Quarter;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

HELP 文 [Interactive SQL]

Interactive SQL 環境のヘルプを提供します。

構文

HELP ['*topic*']

備考

HELP 文を使用して、SQLAnywhere のマニュアルにアクセスします。

ヘルプの *topic* をオプションで指定できます。*topic* は一重引用符で囲む必要があります。一部のヘルプフォーマットでは topic を指定できません。この場合、Interactive SQL の一般的なヘルプページへのリンクが表示されます。

topic には次の値を使用できます。

- SQL Anywhere エラーコード
- SQL 文のキーワード (INSERT、UPDATE、SELECT、CREATE DATABASE など)

パーミッション

なし

関連する動作

なし

参照

- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

IF 文

SQL 文の条件付き実行を制御します。

構文

IF *search-condition* **THEN** *statement-list*
[**ELSEIF** { *search-condition* | *operation-type* } **THEN** *statement-list*] ...
[**ELSE** *statement-list*]
{ **END IF** | **ENDIF** }

備考

IF 文は制御文です。これを使うと *search-condition* が TRUE と評価する SQL 文の最初のリストを条件付きで実行できます。*search-condition* が TRUE と評価せず、ELSE 句が存在する場合、ELSE 句の中の *statement-list* が実行されます。

実行は END IF の後に記述されている最初の文から再開されます。

> **注意**
> IF 文の構文と IF 式の構文を混同しないでください。

パーミッション

なし

関連する動作

なし

参照

- 「BEGIN 文」497 ページ
- 「IF 式」15 ページ
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』
- 「探索条件」34 ページ

標準と互換性

- **SQL/2008**　IF 文は、オプションの SQL/2008 言語機能 P002、"Computational completeness" の一部です。ENDIF キーワードはベンダー拡張です。

例

次のプロシージャーは、IF 文の使い方を示します。

```
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;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
  CLOSE curThisCust;
END;
```

IF 文 [T-SQL]

Watcom SQL IF 文の代わりに、SQL 文の条件付き実行を制御します。

構文

IF *expression statement*
[**ELSE** [**IF** *expression*] *statement*]

備考

Transact-SQL IF と ELSE は、それぞれ単一の SQL 文または複合文 (キーワード BEGIN と END の間) の実行を制御します。

Watcom SQL IF 文と比較すると、Transact-SQL IF 文には THEN がありません。Transact-SQL バージョンには、ELSEIF または END IF キーワードもありません。

パーミッション

なし

関連する動作

なし

標準と互換性

- **SQL/2008** Transact-SQL 拡張。

例

次の例は、Transact-SQL IF 文の使い方を示します。

```
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
```

次の 2 つの文のブロックは、Transact-SQL と Watcom SQL の互換性を示します。

```
/* 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
```

INCLUDE 文 [ESQL]

SQL プリプロセッサーによってスキャンされるソースプログラムにファイルをインクルードします。

構文

INCLUDE *filename*

filename : **SQLDA** | **SQLCA** | *string*

備考

INCLUDE 文は、C プリプロセッサー #include ディレクティブと非常によく似ています。SQL プリプロセッサーは Embedded SQL ソースファイルを読み込んで、Embedded SQL 文をすべて C 言語のソースコードに置き換えます。SQL プリプロセッサーに必要な情報が入ったファイルがある場合は、Embedded SQL INCLUDE 文でそのファイルをインクルードしてください。

2 つのファイル名 SQLCA と SQLDA は特別に認識されます。Embedded SQL ソースファイルでは、Embedded SQL 文の前に次の文が必要です。

```
EXEC SQL INCLUDE SQLCA;
```

この文を C 言語の静的変数宣言を置くことができる場所に入れます。多くの Embedded SQL 文では、SQLCA インクルード文の位置に SQL プリプロセッサーが宣言する変数 (アプリケーション開発者には見えません) が必要となります。SQLDA ファイルを使用している場合は、SQLDA ファイルをインクルードします。

パーミッション

なし

関連する動作

なし

標準と互換性

- **SQL/2008**　ベンダー拡張。

INPUT 文 [Interactive SQL]

外部ファイル、キーボード、ODBC データソース、またはシェイプファイルからデータベーステーブルにデータをインポートします。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 1 - 外部ファイルまたはキーボードからのインポート

INPUT INTO [*owner*.]*table-name input-options*

input-options :
[**(** *column-name*, ... **)**]
[**BY** { **ORDER** | **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**
| **FIXED**]

encoding : *identifier* | *string*

構文 2 - ODBC データソースからのインポート

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* }]

構文 3 - シェイプファイルのロード

INPUT INTO [*owner*.]*table-name*
FROM *filename*
FORMAT SHAPEFILE
[**SRID** *srid-number*
[**ENCODING** *encoding*]

encoding : *identifier* | *string*

パラメーター

BY 句　この句を使用して、リストの順序位置 (デフォルトの ORDER) と名前 (NAME) のどちらを基準として入力ファイルのカラムをテーブルカラムと一致させるかを指定します。すべての入力フォーマットが、ファイルの中にカラム名情報を持っているわけではありません。NAME を使用できるのは、ファイルに名前情報があるフォーマットの場合のみです。

BYTE ORDER MARK 句　この句は、データのバイトオーダーマーク (BOM) を処理するかどうかを指定するときに使用します。

BYTE ORDER MARK 句は、TEXT フォーマットのファイルから読み込む場合にのみ関係します。TEXT 以外の FORMAT 句とともに BYTE ORDER MARK 句を使用すると、エラーが発生します。

BYTE ORDER MARK 句は、UTF-8 か UTF-16 でエンコードされたファイルの読み込みまたは書き込みをするときにのみ使用されます。その他のエンコードで BYTE ORDER MARK 句を使用しようとすると、エラーが発生します。

ENCODING 句が指定されている場合

○ BYTE ORDER MARK オプションが ON のときに、UTF-16BE または UTF-16LE などのエンディアンを持つ UTF-16 エンコードを指定すると、Interactive SQL はデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの検証に使用されます。間違ったエンディアンを指定すると、エラーが返されます。
○ BYTE ORDER MARK オプションが ON のときに、明示的なエンディアンのない UTF-16 エンコードを指定すると、Interactive SQL はデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの確認に使用されます。BOM がない場合は、オペレーティングシステムのエンディアンであると想定されます。
○ BYTE ORDER MARK オプションが ON のときに、UTF-8 エンコードを指定すると、Interactive SQL はデータの最初で BOM を検索します。BOM がある場合は無視されます。

ENCODING 句が指定されていない場合

○ ENCODING 句を指定しないで BYTE ORDER MARK オプションを ON にすると、Interactive SQL は入力データの最初で BOM を検索します。BOM が見つかると、BOM のエンコード (UTF-16BE、UTF-16LE、または UTF-8) に基づいてソースエンコードが自動的に選択され、BOM はロードするデータの一部であるとは見なされなくなります。
○ ENCODING 句を指定せず、BYTE ORDER MARK オプションが OFF であるか、または BOM が入力データの最初で見つからない場合は、クライアントコンピューターのエンコードが使用されます。

COLUMN WIDTHS 句　COLUMN WIDTHS 句で入力ファイルのカラムの幅を指定しますが、これは FIXED フォーマットの場合のみ指定できます。COLUMN WIDTHS を指定しない場合、幅はデータベースカラムのタイプによって決定されます。

CREATE TABLE 句　CREATE TABLE 句を使用して、インポート先テーブルが存在しない場合に作成するかどうかを指定します。デフォルトは ON です。

DELIMITED BY 句　DELIMITED BY 句を使用すると、TEXT 入力フォーマットのカラム値間でデリミターとして使用する文字列を指定できます。デフォルトのデリミターはカンマです。

ENCODING 句　*encoding* 引数では、ファイルの読み込みに使用されるエンコードを指定します。ENCODING 句は、TEXT および SHAPEFILE フォーマットでのみ使用できます。

Interactive SQL を実行するとき、データのインポートに使用されるエンコードは次の順序で決定されます。

○ENCODING 句で指定されたエンコード (この句が指定されている場合)。

○default_isql_encoding オプションで指定されたエンコード (このオプションが設定されている場合)。

○ファイルにバイトオーダーマークがある場合は、適切な Unicode エンコードが使用されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○実行しているコンピューターのデフォルトのエンコード。英語版 Windows コンピューターでは、デフォルトエンコードは 1252 または 850 です。

OUTPUT 文を使用して入力ファイルを作成したときにエンコードを指定した場合は、INPUT 文でも同じ ENCODING 句を指定してください。

ESCAPE CHARACTER 句　16 進コードに使用するデフォルトのエスケープ文字は円記号 (¥) です。たとえば、¥x0A は改行文字です。

改行文字は ¥n と指定できます。他の文字は、タブ文字は ¥x09 などのように、16 進の ASCII コードを使用して指定できます。2 つの円記号 (¥) は 1 つの円記号として解釈されます。円記号 (¥) の後に n、x、X、¥ 以外の文字がある場合、それらは別々の文字と解釈されます。たとえば、¥q は円記号と q と解釈されます。

エスケープ文字は、ESCAPE CHARACTER 句を使って変更することができます。たとえば、感嘆符 (!) をエスケープ文字として使用するには、次のように指定します。

```
... ESCAPE CHARACTER '!'
```

ESCAPES 句　ESCAPES を ON (デフォルト) にすると、データベースサーバーによって円記号に続く文字は特殊文字として解釈されます。ESCAPES を OFF にすると、ソースに記載されているとおりに文字が読み取られます。

FORMAT 句　FORMAT 句によって、出力ファイルフォーマットを指定します。

FORMAT 句が指定されていない場合、input_format オプションでフォーマットが設定されます。デフォルト設定は TEXT です。Interactive SQL で **[ツール]** » **[オプション]** » **[インポート／エクスポート]** をクリックし、**[デフォルトのインポートフォーマット]** フィールドで **[テキスト]** または **[固定]** を指定しても入力フォーマットを指定できます。

サポートされるフォーマットは次のとおりです。

○ **TEXT**　入力行は文字であり、1 行あたり 1 つのローで構成され、カラム値はデリミターで区切られているものと見なされます。アルファベットの文字列を一重引用符または二重引用符で囲むことができます。デリミターを含む文字列は、一重引用符または二重引用符のどちらかで囲みます。文字列そのものに一重引用符または二重引用符を含める場合は、引用符文字を文字列の中で 2 つ続けて入力して使用してください。DELIMITED BY 句を使って、デフォルトのカンマではなく、他のデリミターを使用できます。

他の 3 つの特別なシーケンスも認識できます。2 つの文字 ¥n は改行文字を表し、¥¥ は 1 つの (¥) を表し、シーケンス ¥xDD は 16 進コード DD の文字を表します。

省略された値は NULL として扱われます。その位置の値を NULL にできない場合は、数値カラムには 0、文字カラムには空の文字列が挿入されます。

○ **FIXED**　入力行は固定フォーマットです。カラムの幅は、COLUMN WIDTHS 句を使って指定できます。カラムの幅を指定しない場合、ファイル内のカラム幅は、対応するデータベースカラムで許容される文字の最大値と同じ値にしてください。

FIXED フォーマットは、埋め込まれた改行とファイルの終わり文字シーケンスを含むバイナリカラムでは使用できません。

○ **SHAPEFILE [SRID *srid-number*]**　入力は、ESRI シェイプファイルの形式です。LOAD 文と違って、INPUT 文を使用してシェイプファイルをロードする場合、シェイプファイルをクライアントコンピューターに置く必要があります。関連付けられている *.shx* ファイルと *.dbf* ファイルは、*.shp* ファイルと同じディレクトリに配置し、ベースファイル名を同じ名前にしてください。

シェイプファイルをロードするときにエンコードが指定されていない場合、デフォルトは ISO-8859-1 になります。

SRID 句を使用して、ジオメトリに関連付ける SRID を指定します。SRID を指定しない場合、デフォルトとして SRID 0 が使用されます。理想としては、シェイプファイルのプロジェクトファイル (*.prj*) に記載されている SRID と同じ SRID にしてください。この SRID を使用できない場合は、これと同等の SRID を使用します。SQL Anywhere には、sa_install_feature システムプロシージャーを使用してデータベースに追加できる SRID が多数用意されています。

DBASE II、DBASE III、FoxPro、Lotus 123、または Excel 97 など、その他のフォーマットを使用する場合は、INPUT USING を使用する必要があります。

FROM *filename* 句　*file-name* は引用符で囲むことも囲まないこともできます。文字列を引用符で囲む場合、他の SQL 文字列と同じフォーマット要件に従います。

ディレクトリパスを示すには、円記号 (¥) を 2 つの円記号で表してください。ファイル *c:¥temp¥input.dat* から Employees テーブルにデータをロードする文は、次のようになります。

```
INPUT INTO Employees
FROM 'c:¥¥temp¥¥input.dat';
```

相対的な *filename* のロケーションは、次のように決定されます。

○Interactive SQL で INPUT 文を直接実行した場合、*filename* へのパスは、Interactive SQL が動作しているディレクトリとの相対パスとして解決されます。たとえば、ディレクトリ *c:¥work* から Interactive SQL を開き、次の文を実行すると想定します。

```
INPUT INTO Employees
 FROM 'inputs¥¥inputfile.dat';
```

Interactive SQL は *c:¥work¥inputs¥inputfile.dat* を検索します。

○Interactive SQL では、*.sql* ファイルに INPUT 文が存在する場合、最初に *filename* へのパスをこのファイルのロケーションとの相対パスとして解決しようとします。これが失敗した場合は、Interactive SQL が動作しているディレクトリとの相対パスで *filename* を検索します。

たとえば、ファイル *c:¥homework¥inputs.sql* があり、この中に次の文が含まれていると想定します。

```
INPUT INTO Employees
 FROM 'inputs¥¥inputfile.dat';
```

Interactive SQL は、最初に *c:¥homework¥inputs* で *inputfile.dat* を検索します。このロケーションで *inputfile.dat* が見つからない場合は、Interactive SQL が動作しているディレクトリを検索します。

FROM *source-table-name* 句　*source-table-name* パラメーターは、送信元データベースのテーブル名を含む、引用符で囲まれた文字列です。この名前は、*database-name.owner.table-name*、

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

owner.table-name、または *table-name* の形式で指定できます。ピリオドが送信元データベースのネイティブのセパレーターではない場合も、コンポーネントの区切りにはピリオドを使用します。送信元データベースにデータベース名は必要であるが所有者名は不要な場合、*source-table-name* のフォーマットは *database..table* とする必要があります (この場合、所有者名は空)。パラメーター内の名前を引用符で囲まないでください。たとえば、'dba."my-table"' を使用しないで、代わりに 'dba.my-table' を使用します。

INTO 句　データを入力するテーブルの名前です。

PROMPT 句　PROMPT 句では、ユーザーがロー内の各カラムの値を入力できます。ウィンドウモードで実行している場合は、ユーザーが新規ローの値を入力できるウィンドウが表示されます。コマンドラインから Interactive SQL を実行している場合は、コマンドラインに各カラムの値を入力するように求められます。

NOSTRIP 句　通常、TEXT 入力フォーマットの場合、引用符の付いていない文字列から後続ブランクを削除してから値を挿入します。NOSTRIP を使うと、後続ブランクを削除しないようにできます。後続ブランクは、オプションが使用されているかどうかにかかわらず、引用符付きの文字列からは削除されません。先行ブランクは、NOSTRIP オプション設定にかかわらず引用符の付いていない文字列から削除されます。

SKIP 句　TEXT ファイルから行を挿入する場合、SKIP 句はファイルの先頭から指定の行数を省略します。指定する数は正の整数にします。SKIP 句を使用できるのは、TEXT フォーマットの場合のみです。

ファイル内の行数を超える行数を指定した場合、INPUT 文はデータを一切挿入せず、エラーも返されません。

USING 句　USING 句は ODBC データソースからデータを入力します。DSN オプションを使用して ODBC データソース名を指定するか、または DRIVER オプションを使用して ODBC ドライバー名と接続パラメーターを指定できます。*connection-parameter* パラメーターは、データベース固有の接続パラメーターのリストです。

odbc-data-source パラメーターは、ユーザー名または ODBC データソース名です。たとえば、SQL Anywhere サンプルデータベースの *odbc-data-source* は SQL Anywhere 12 Demo です。

odbc-driver-name パラメーターは ODBC ドライバー名です。SQL Anywhere データベースでは、*odbc-driver-name* は SQL Anywhere 12 です。Ultra Light データベースでは、*odbc-driver-name* は Ultra Light 12 です。

備考

INPUT 文を使用すると、指定したデータベーステーブルの中に効率よく大量に挿入できます。入力行は、入力ウィンドウを使用してユーザーが入力するか (PROMPT を指定した場合)、またはファイルから読み込まれます (FROM *filename* を指定した場合)。どちらも指定しない場合は、INPUT 文を含む SQL スクリプトファイルから入力が読み込まれます。Interactive SQL の場合は、**[SQL 文]** ウィンドウ枠から直接読み込むこともできます。この場合、入力は、文字列 END のみを含む行で終了します。

[SQL 文] ウィンドウ枠から入力を直接読み込む場合、INPUT 文の末尾に挿入するレコード値の前に、セミコロンを指定する必要があります。次に例を示します。

```
INPUT INTO Owner.TableName;
value1, value2, value3
value1, value2, value3
value1, value2, value3
value1, value2, value3
END
```

END キーワード (セミコロンなし) は、ファイル名が指定されておらず、PROMPT キーワードを含まない INPUT 文のデータを終了します。

カラムリストを指定すると、目的のテーブルの指定したカラムにデータが挿入されます。デフォルトで、INPUT 文では、入力ファイルのカラム値がデータベーステーブル定義のカラム値と同じ順序で出現していると想定されます。入力ファイルのカラムの順序が異なる場合は、INPUT 文の末尾に実際のカラムの順序をリストしてください。

たとえば、次の文を使用してテーブルを作成するとします。

```
CREATE TABLE inventory (
Quantity INTEGER,
item VARCHAR(60)
);
```

また、quantity 値の前に name 値がある入力ファイル *stock.txt* から TEXT データをインポートするとします。

```
'Shirts', 100
'Shorts', 60
```

この場合は、データが適切に挿入されるように、INPUT 文の末尾に入力ファイルの実際のカラム順序をリストします。

```
INPUT INTO inventory
FROM stock.txt
FORMAT TEXT
(item, Quantity);
```

デフォルトでは、INPUT 文は、エラーの原因となるローを挿入しようとすると停止します。on_error と conversion_error オプションを設定すると、さまざまな方法でエラーを処理できます。

文字列値が INPUT でトランケートされる場合、Interactive SQL は **[メッセージ]** タブに警告を表示します。NOT NULL カラムで値が見つからない場合、数値型カラムは 0 に設定され、数値型でないカラムは空の文字列に設定されます。INPUT で NULL のローを挿入しようとすると、入力ファイルには空のローができます。

INPUT 文は Interactive SQL コマンドであるため、サーバーによって実行される SQL Anywhere 文や Ultra Light 文では使用できません。

パーミッション

テーブルまたはビューに対する INSERT パーミッションが必要です。

関連する動作

なし

参照

- 「INPUT 文を使用したデータのインポート」『SQL Anywhere サーバー SQL の使用法』
- 「データインポート」『SQL Anywhere サーバー SQL の使用法』
- 「新しい Ultra Light データベースへの ASCII データのインポート」『Ultra Light データベース管理とリファレンス』
- 「OUTPUT 文 [Interactive SQL]」848 ページ
- 「INSERT 文」802 ページ
- 「SET OPTION 文 [Interactive SQL]」916 ページ
- 「LOAD TABLE 文」814 ページ
- BYTE ORDER MARK 句、INPUT 文 795 ページ
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』
- 「アンロードユーティリティ (dbunload)」『SQL Anywhere サーバー データベース管理』
- 「ESRI シェイプファイルのサポート」『SQL Anywhere サーバー 空間データサポート』
- 「サポートされている文字セット」『SQL Anywhere サーバー データベース管理』
- 「プロシージャー、トリガー、イベント、バッチで使用できる文」『SQL Anywhere サーバー SQL の使用法』
- 「チュートリアル：空間機能の実験」『SQL Anywhere サーバー 空間データサポート』
- 「input_format オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「sa_install_feature システムプロシージャー」1106 ページ
- 「default_isql_encoding オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL の結果セットからデータベースにローを挿入する」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次は、TEXT ファイルからの INPUT 文の例です。

```
INPUT INTO Employees
  FROM new_emp.inp
  FORMAT TEXT;
```

次の例では、テーブル ulTest をテーブル saTest にコピーします。ulTest は Ultra Light データベースのファイル *C:¥test¥myULDatabase.udb* 内のテーブルです。saTest は *demo.db* に作成されたテーブルです。

```
INPUT USING 'driver=UltraLite 12;dbf=C:¥¥test¥¥myULDatabase.udb'
  FROM "ulTest" INTO "saTest";
```

次の例では、架空のシェイプファイル *myshapefile.shp* を myTable テーブルにロードし、SRID 4269 をシェイプファイルのジオメトリに割り当てます。

```
INPUT INTO myTable
  FROM 'myshapefile.shp'
  FORMAT SHAPEFILE SRID 4269
```

INSERT 文

1 つのロー (構文 1) またはデータベースのある場所から選択したロー (構文 2) をテーブルに挿入します。

構文 1

INSERT [**INTO**] [*owner*.]*table-name* [**(** *column-name*, ... **)**]
[**ON EXISTING** {
  ERROR
  | **SKIP**
  | **UPDATE** [**DEFAULTS** { **ON** | **OFF** }]
  }]
{ **DEFAULT VALUES**
  | **VALUES** *row-value-constructor* [, ...] }
[**OPTION(** *query-hint* [, ...] **)**]

構文 2

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 (許可されたインジケーター)
| *string*
| *identifier*
| *number*

insert-expression :
expression | **DEFAULT**

row-value-constructor :
([*insert-expression* [, ...]]**)**

パラメーター

VALUES 句　VALUES 句は、挿入する値を指定するときに使用します。対象となるカラムに定義されているデフォルト値を挿入する場合は、DEFAULT VALUES を指定します。VALUES () と指定しても DEFAULT VALUES と同義になります。また、VALUES 句では、1 つの文に複数

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

のローの値を挿入できるように、ロー値コンストラクターがサポートされています。各 *row-value-constructor* 内の *insert-expression* 値の数と順序は、INTO 句で指定したカラムリストに対応させてください。カラムリストを指定していない場合は、テーブルの完全な順序のカラムリストであると見なされます。空のカラムリスト () を指定する場合は、テーブルの各カラムにデフォルト値が含まれている必要があります。

ローの挿入時にエラーが発生すると、すべての変更がロールバックされます。

WITH AUTO NAME 句　WITH AUTO NAME は、構文 2 にのみ適用されます。WITH AUTO NAME を指定すると、クエリブロックの項目名によってデータの所属カラムが決まります。クエリブロックの項目には、カラム参照かエイリアスの式を指定してください。クエリブロックで定義されていない送信先カラムには、デフォルト値が割り当てられます。これは送信先テーブルのカラム数が多い場合に役立ちます。

WITH AUTO NAME 句を指定し、ターゲットテーブルのカラムと一致しないカラムがクエリブロックに含まれている場合、INSERT 文はエラーを返します。たとえば、次の文を実行すると、SELECT クエリブロックの操作カラムが MyTable5 テーブルのどのカラムとも一致しないことを示すエラーが返されます。

```
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 句　INSERT 文の ON EXISTING 句は、両方の構文に適用されます。ON EXISTING 句は、プライマリキールックアップに基づいて、新しいカラム値でテーブルの既存のローを更新します。この句は、プライマリキーが設定されたテーブルでのみ使用できます。プライマリキーがないテーブルでこの句を使用すると、構文エラーになります。ON EXISTING を指定してプロキシテーブルに値を挿入することはできません。

注意
ON EXISTING 条件を満たすローが数多く存在する可能性がある場合は、代わりに MERGE 文を使用してください。MERGE 文を使用すると、一致するローに対して実行するアクションをより詳細に制御できます。また、一致を定義するためのより高度な構文も使用できます。

ON EXISTING 句を指定すると、データベースサーバーは各入力ローに対してプライマリキールックアップを実行します。対応するローがテーブルに存在しなければ、新しいローを挿入します。ローがテーブルにすでに存在する場合は、入力ローを無視する (SKIP)、重複したキー値のエラーメッセージを生成する (ERROR)、入力ローの値を使用して古い値を更新する (UPDATE) 操作のいずれかを選択できます。ON EXISTING 句を指定しなかった場合は、デフォルトでは、ローがすでに存在するテーブルにローを挿入しようとすると、キー値の重複エラーになります。また、ON EXISTING ERROR 句を指定するときと同じです。スキップされるローは、@@rowcount 変数に含まれます。

ON EXISTING UPDATE を使用している場合、入力したローは格納されているローと比較されます。入力ローに指定されているすべてのカラム値は、格納されているローの対応するカラム値を

置換します。同様に、入力ローに明示的にカラム値が指定されていない場合、格納されているローの対応するカラム値は変化しません。ただし、デフォルト値があるカラムの場合は例外です。また、デフォルト値があるカラム (DEFAULT AUTOINCREMENT カラムなど) を含む ON EXISTING UPDATE 句を使用する場合、ON EXISTING UPDATE DEFAULTS ON を指定することでデフォルト値でカラム値を更新するか、ON EXISTING UPDATE DEFAULTS OFF を指定してカラム値をそのままにすることができます。何も指定しない場合、デフォルトの動作は ON EXISTING UPDATE DEFAULTS OFF です。

注意
DEFAULTS ON パラメーターと DEFAULTS OFF パラメーターは、DEFAULT TIMESTAMP、DEFAULT UTC TIMESTAMP、または DEFAULT LAST USER の値に関係ありません。これらのカラムでは、格納されているローの値は UPDATE 中に常に更新されます。

ON EXISTING SKIP 句と ON EXISTING ERROR 句を使用するときにテーブルにデフォルトのカラムが含まれていると、サーバーは、すでに存在するローに対しても、デフォルト値を計算します。結果として、AUTOINCREMENT のようなデフォルト値が、スキップされたローに対しても影響を及ぼします。AUTOINCREMENT の場合は、AUTOINCREMENT のシーケンスで値がスキップされます。次の例で説明します。

```
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 );
```

最初の INSERT 文に定義されたローが挿入され、c2 は 1 に設定されます。2 番目の INSERT 文に定義されたローは、既存のローに一致するため、スキップされます。ただし、オートインクリメントカウンターは 2 に増分されたまま残ります (既存のローには影響ありません)。3 番目の INSERT 文に定義されたローが挿入され、c2 の値は 3 に設定されます。したがって、上記の例で挿入された値は次のようになります。

```
20,1
30,3
```

警告
SQL Remote を使用している場合、DEFAULT LAST USER カラムをレプリケートしないでください。カラムをレプリケートすると、カラム値はレプリケートされた値ではなく、SQL Remote ユーザーに設定されます。

OPTION 句　この句は、文を実行するためのヒントを指定するときに使用します。次のヒントがサポートされます。

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- *option-name* = *option-value* クエリテキスト内の OPTION(isolation_level = ...) の指定は、クエリの独立性レベルを指定する他のいずれの手段よりも優先されることに注意してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

INSERT 文を使って、データベーステーブルに新しいローを追加します。

テキストインデックスとマテリアライズドビューは基本となるテーブルデータの変更によって影響を受けるため、基本となるテーブルにデータをバルクロード (LOAD TABLE、INSERT、MERGE) する前に、従属するテキストインデックスまたはマテリアライズドビューをトランケートすることを検討してください。

- **構文 1**　指定された式のカラム値を使用して、1 つのローまたは複数のローを挿入します。複数のローを指定する場合は、追加のカッコで区切ります。キーワード DEFAULT を使うと、カラムのデフォルト値を挿入できます。オプションのカラム名のリストを指定すると、指定したカラムの中に値が 1 つずつ挿入されます。カラム名のリストを指定しないと、作成順 (SELECT * を使って取り出すときと同じ順序) に値がテーブルカラムの中に挿入されます。ローは、テーブル内の任意の位置に挿入されます。リレーショナルデータベースでは、テーブルの順序は指定されません。

- **構文 2**　一般的な SELECT 文の結果を使用して、テーブルに大量に挿入します。SELECT 文に ORDER BY 句が含まれていない場合、任意の順序で挿入が行われます。

カラム名を指定すると、SELECT リストのカラムは、カラムリストに指定されたカラムまたは作成順に並べたカラムと順に一致します。

ビューを定義しているクエリ指定が更新可能な場合は、ビューに対して挿入操作を実行できます。

テーブルに挿入した文字列は、データベースが大文字と小文字を区別するかどうかに関係なく、常に入力された大文字と小文字の状態のままで格納されます。そのため、テーブルに挿入される文字列 Value は、常に V が大文字、残りの文字が小文字でデータベースに格納されます。SELECT 文は、文字列を Value として返します。ただし、データベースで大文字と小文字が区別されない場合は、すべての比較において Value は value、VALUE などと同じと見なされます。さらに、単一カラムのプライマリキーにエントリ Value がある場合は、プライマリキーがユニークでなくなるので、INSERT 文による value の追加は拒否されます。

INSERT 文を使用してデータを大量に挿入する場合も、カラム統計は更新されます。

> **注意**
> テーブルに多数のローを挿入するには、個別の INSERT 文を多数実行するよりも、可能であればカーソルを宣言し、そのカーソルを通じてローを挿入する方が効率的です。データを挿入する前に、テーブルページごとに今後の更新のために確保する空き領域の割合を指定できます。

パーミッション

テーブルに対する INSERT パーミッションが必要です。

ON EXISTING UPDATE 句が指定された場合、テーブルの UPDATE パーミッションも必要になります。

関連する動作

なし

参照

- @@rowcount グローバル変数 82 ページ
- 「データインポート」『SQL Anywhere サーバー SQL の使用法』
- 「通常のビュー」『SQL Anywhere サーバー SQL の使用法』
- 「ALTER TABLE 文」462 ページ
- 「MERGE 文」834 ページ
- 「INPUT 文 [Interactive SQL]」794 ページ
- 「LOAD TABLE 文」814 ページ
- 「UPDATE 文」976 ページ
- 「SELECT 文」895 ページ
- 「TRUNCATE 文」961 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「DELETE 文」697 ページ
- 「PUT 文 [ESQL]」860 ページ
- 「クライアントコンピューター上のデータへのアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「INSERT を使用したデータの追加」『SQL Anywhere サーバー SQL の使用法』
- OPTION 句、SELECT 文902 ページ

標準と互換性

- **SQL/2008**　INSERT 文は、SQL/2008 標準のコア機能です。DEFAULT VALUES 句は、オプションの SQL 言語機能 F222、"INSERT statement: DEFAULT VALUES clause" です。INSERT 文でのロー値コンストラクターのサポートは、オプションの SQL 言語機能 F641、"Row and table constructors" の一部です。VALUES キーワードはベンダー拡張であり、SQL Anywhere で挿入する式のリストを指定する場合に必須です。ただし、VALUES は SQL/2008 には含まれていません。

  いくつかのオプションの構成はベンダー拡張です。これらのベンダー拡張を次に示します。

  ○INSERT...ON EXISTING 句はベンダー拡張です。多くの場合、SQL/2008 準拠の機能では、MERGE 文が同義となります。

  ○OPTION 句。

  ○WITH AUTO NAME 句。

例

データベースに Eastern Sales 部を追加します。

```
INSERT
INTO Departments ( DepartmentID, DepartmentName )
VALUES ( 230, 'Eastern Sales' );
```

テーブル DepartmentHead を作成し、WITH AUTO NAME 構文を使用して部長とその部の名前を挿入します。

```
CREATE TABLE DepartmentHead(
    pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
    DepartmentName VARCHAR(128),
    ManagerName VARCHAR(128) );
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
INSERT
INTO DepartmentHead WITH AUTO NAME
SELECT GivenName || ' ' || Surname AS ManagerName,
    DepartmentName
FROM Employees JOIN Departments
ON EmployeeID = DepartmentHeadID;
```

テーブル MyTable5 を作成し、WITH AUTO NAME 構文を使用してデータを移植します。

```
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
FROM SYS.SYSTAB t
WHERE table_id <= 10;
```

データベースの現在の独立性レベル設定ではなく、独立性レベル 3 で文を実行し、新しい部署を挿入します。

```
INSERT INTO Departments
  (DepartmentID, DepartmentName, DepartmentHeadID)
  VALUES(600, 'Foreign Sales', 129)
  OPTION( isolation_level = 3 );
```

次の例では、テーブルに 3 つのローを挿入します。

```
INSERT INTO T (c1,c2,c3)
VALUES (1,10,100), (2,20,200), (3,30,300);
```

次の例では、INSERT 文は、各カラムにデフォルト値が含まれる 4 つのカラムのテーブルに 3 つのローを挿入します。

```
INSERT INTO T ()
VALUES (), (), ();
```

INSTALL EXTERNAL OBJECT 文

外部環境で実行できるオブジェクトをインストールします。

構文

INSTALL EXTERNAL OBJECT *object-name*
[*update-mode*]
FROM { **FILE** *file-path* | **VALUE** *expression* }
ENVIRONMENT *environment-name*

environment-name :
PERL
| **PHP**

update-mode :
NEW
| **UPDATE**

パラメーター

object-name データベース内で、インストールしたオブジェクトを識別する名前。

update-mode オブジェクトの更新モード。更新モードを省略すると、NEW が使用されます。

file-path オブジェクトのインストール元となる、サーバーコンピューターのロケーション。

environment-name 外部オブジェクトが実行される外部環境の名前。

備考

外部環境の詳細については、「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』を参照してください。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』
- 「ALTER EXTERNAL ENVIRONMENT 文」420 ページ
- 「REMOVE EXTERNAL OBJECT 文」875 ページ
- 「START EXTERNAL ENVIRONMENT 文」939 ページ
- 「STOP EXTERNAL ENVIRONMENT 文」947 ページ
- 「SYSEXTERNENV システムビュー」1265 ページ
- 「SYSEXTERNENVOBJECT システムビュー」1266 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

この例では、ファイルに格納されている Perl スクリプトをデータベースにインストールします。

```
INSTALL EXTERNAL OBJECT 'PerlScript'
NEW
FROM FILE 'perlfile.pl'
ENVIRONMENT PERL;
```

また、次のように、Perl コードを式から構築してインストールできます。

```
INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE 'sub WriteToServerConsole { print $sa_output_handle $_[0]; }'
ENVIRONMENT PERL;
```

また、次のように、Perl コードを変数から構築してインストールできます。

```
CREATE VARIABLE PerlVariable LONG VARCHAR;
SET PerlVariable =
```

```
'sub WriteToServerConsole { print $sa_output_handle $_[0]; }';

INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE PerlVariable
ENVIRONMENT PERL;
```

INSTALL JAVA 文

Java クラスをデータベース内で使用できるようにします。

構文

INSTALL JAVA
[**NEW** | **UPDATE**]
[**JAR** *jar-name*]
FROM { **FILE** *filename* | *expression* }

パラメーター

NEW キーワードと UPDATE キーワードの句　NEW のインストールモードを指定する場合、参照する Java クラスは現在インストールされているクラスの更新ではなく、新しいクラスにしてください。データベース内に同じ名前のクラスがあり、NEW インストールモードが使用されている場合は、エラーになります。

UPDATE を指定すると、参照する Java クラスは指定されたデータベースにすでにインストールされている Java クラスの代替を含むことがあります。

install-mode が省略されると、デフォルトは NEW です。

JAR 句　この句を指定する場合は、*filename* に jar ファイルを指定します。JAR ファイルは、通常 *.jar* または *.zip* の拡張子を持ちます。

インストールされた jar と zip ファイルは、圧縮または圧縮解除できます。圧縮スキームには違いがあるため、テキストリソースを含む jar は圧縮をオフにした状態で作成することを強くおすすめします。

JAR オプションを指定すると、jar の持つクラスがインストールされた後で jar として保持されます。この jar は、これらのクラスの jar に関連付けられています。JAR オプションと一緒にデータベースにインストールされている jar は、データベースの保持された jar と呼ばれます。

jar-name は、255 バイト以内の文字列値です。後続の INSTALL JAVA 文、UPDATE 文、REMOVE JAVA 文で保持された jar を識別するために、*jar-name* が使用されます。

FROM FILE 句　インストールする Java クラスのロケーションを指定します。

filename でサポートされるフォーマットは、*'c:¥libs¥jarname.jar'* や *'/usr/u/libs/jarname.jar'* のような完全に修飾されたファイル名と、データベースサーバーの現在の作業ディレクトリを基準にした相対ファイル名です。

filename には、クラスファイルまたは jar ファイルを指定してください。

FROM 句　式は、値として有効なクラスファイルまたは jar ファイルを持つバイナリ型に評価されるようにします。

備考

各クラスのクラス定義は、最初にそのクラスを使用するときに各接続の VM でロードされます。クラスをインストールすると、接続の VM が暗黙的に再起動されます。このため、INSTALL が NEW または UPDATE のいずれの *install-mode* を持つかにかかわらず、新しいクラスに即時アクセスします。VM が再起動されるため、Java 静的変数に格納されていた値はすべて失われ、Java クラスタイプの SQL 変数はすべて削除されます。

他の接続では、新しいクラスは VM が最初にクラスにアクセスして次のときにロードされます。VM がそのクラスをロード済みの場合、その接続で VM を再起動するまで、その接続から新しい接続は見えません。

パーミッション

INSTALL JAVA 文を実行するには、DBA 権限が必要です。

すべてのインストールされたクラスは、すべてのユーザーがすべての方法で参照できます。

Windows Mobile ではサポートされません。

参照

- 「REMOVE JAVA 文」876 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、クラスのファイル名とロケーションを指定して、ユーザーの作成した Demo という名前の Java クラスをインストールします。

```
INSTALL JAVA NEW
FROM FILE 'D:¥JavaClass¥Demo.class';
```

次の文は、zip ファイルに含まれるすべてのクラスをインストールし、それらを JAR ファイル名でデータベース内に関連付けます。

```
INSTALL JAVA
JAR 'Widgets'
FROM FILE 'C:¥Jars¥Widget.zip';
```

ここでも、zip ファイルのロケーションは保持されません。クラスは完全に修飾されたクラス名 (パッケージ名とクラス名) を参照しなければなりません。

INTERSECT 文

2 つ以上のクエリの結果セット間の共通部分を計算します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

[**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 : 「一般的な SQL 構文要素」403 ページ を参照。

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* **=** *option-value*

option-name : *identifier*

option-value :
hostvar (許可されたインジケーター)
| *string*
| *identifier*
| *number*

パラメーター

FOR XML 句　FOR XML 句の詳細については、「SELECT 文」895 ページを参照してください。

OPTION 句　この句は、文を実行するためのヒントを指定するときに使用します。次のヒントがサポートされます。

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- *option-name* = *option-value* クエリテキスト内の OPTION(isolation_level = ...) の指定は、クエリの独立性レベルを指定する他のいずれの手段よりも優先されることに注意してください。

備考

INTERSECT は、2 つのクエリブロックの結果セット間の集合積を計算します。クエリブロックはネストでき、ネストされた SELECT 文や、集合演算子 UNION、EXCEPT、または INTERSECT を含めることができます。INTERSECT の単独での指定は、INTERSECT DISTINCT の指定と同義です。

INTERSECT ALL では、集合積ではなく、バッグ積を実装します。たとえば、最初の *query-block* 内に特定の値を持つ 5 つの (重複) ローがあり、2 番目の *query-block* 内に同一の値を持つ 3 つの重複ローがある場合、INTERSECT ALL は 3 つのローを返します。

query-block 内に重複ローがない場合、INTERSECT の結果は INTERSECT ALL と同じになります。

2 つの *query-block* の結果セットは和両立する必要があります。それぞれの SELECT リストには同数の項目が含まれており、かつ、各項目の型は比較可能である必要があります。2 つの SELECT リスト内の対応する項目が異なるデータ型の場合、SQL Anywhere は結果の中から対応するカラムのデータ型を選択し、各 *query-block* のカラムを自動的にそれぞれ変換します。

表示されるカラム名は最初の *query-block* に表示されるカラム名と同じであり、これらの名前は ORDER BY 句と一致する式の名前を判断するために使用されます。結果セットのカラム名をカスタマイズする別の方法として、共通のテーブル式 (WITH 句) を使用する方法があります。

パーミッション

各 *query-block* には SELECT パーミッションが必要です。

関連する動作

なし

参照

- 「EXCEPT 文」738 ページ
- 「UNION 文」964 ページ
- 「SELECT 文」895 ページ
- OPTION 句、SELECT 文902 ページ

標準と互換性

- **SQL/2008**　INTERSECT は、SQL/2008 標準のオプションの SQL 言語機能 F302 です。INTERSECT とともに DISTINCT キーワードを明示的に指定する方法は、オプションの SQL 言語機能 T551 です。INTERSECT とともに ORDER BY 句を指定する方法は、SQL 言語機能 F850 です。ORDER BY 句を含む *query-block* は、SQL/2008 機能 F851 の構成要素です。ロー制限句 (SELECT TOP または LIMIT) を含むクエリブロックは、コンテキストに応じて、オプションの SQL 言語機能 F857 または F858 を構成します。FOR XML 句と OPTION 句はベンダー拡張です。

- **Transact-SQL**　INTERSECT は、Adaptive Server Enterprise ではサポートされていません。ただし、SQL Anywhere でサポートされている Transact-SQL ダイアレクトでは、INTERSECT ALL と INTERSECT DISTINCT の両方を使用できます。

例

INTERSECT の使用例については、「集合演算子と NULL」『SQL Anywhere サーバー SQL の使用法』を参照してください。

LEAVE 文

複合文またはループから出ます。

構文

LEAVE *statement-label*

備考

LEAVE 文は制御文です。これを使うと、指定したラベルの複合文または指定したラベルのループを離れることができます。実行は、複合文またはループの後に記述されている最初の文から再開されます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

プロシージャーまたはトリガーの本文である複合文は、プロシージャーまたはトリガーの名前と同じ暗黙のラベルを持っています。

パーミッション

なし

関連する動作

なし

参照

- 「LOOP 文」832 ページ
- 「FOR 文」753 ページ
- 「BEGIN 文」497 ページ
- 「ストアドプロシージャー、トリガー、バッチ、ユーザー定義関数」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　LEAVE 文は、オプションの SQL/2008 言語機能 P002、"Computational completeness" の一部です。

例

次のフラグメントは、LEAVE 文を使ってループを離れる方法を示します。

```
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
```

次のフラグメントは、ネストされたループ内で LEAVE を使用します。

```
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
```

LOAD STATISTICS 文

内部でのみ使用。統計情報を ISYSCOLSTAT システムテーブルにロードします。dbunload ユーティリティで、古いデータベースからカラム統計をアンロードするために使用されます。手動では使用しないでください。

構文

LOAD STATISTICS [[*owner*.]*table-name*.]*column-name*
format-id, *density*, *max-steps*, *actual-steps*, *step-values*, *frequencies*

パラメーター

format-id ISYSCOLSTAT システムテーブルにある残りのローのフォーマットを決定するための内部フィールド。

density カラムの単一の値の加重平均による選択性の推定。ローに格納されている大きい単一値の選択性は考慮されません。

max-steps ヒストグラム内で可能な最大ステップ数。

actual-steps この時点で実際に使用されているステップ数。

step-values ヒストグラムのステップの境界値。

frequencies ヒストグラムのステップの選択性。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「ISYSCOLSTAT システムテーブル」995 ページ
- 「アンロードユーティリティ (dbunload)」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

LOAD TABLE 文

外部ファイルからデータベーステーブルにバルクデータをインポートします。

構文

LOAD [**INTO**] **TABLE** [*owner*.]*table-name*
[**(** *column-name*, ... **)**]

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

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 :
BYTE ORDER MARK { **ON** | **OFF** }
| **CHECK CONSTRAINTS** { **ON** | **OFF** }
| { **COMPRESSED** | **AUTO COMPRESSED** | **NOT COMPRESSED** }
| { **ENCRYPTED KEY '***key***'** | **NOT ENCRYPTED** }
| **COMMENTS INTRODUCED BY** *comment-prefix*
| **COMPUTES** { **ON** | **OFF** }
| **DEFAULTS** { **ON** | **OFF** }
| **DELIMITED BY** *string*
| **ENCODING** *encoding*
| **ESCAPE CHARACTER** *character*
| **ESCAPES** { **ON** | **OFF** }
| **FORMAT** {
  TEXT
  | **BCP**
  | **XML** *row-xpath* **(** *column-xpath*,... **)** [**NAMESPACES** *namespace*] }
  | **SHAPEFILE**
| **HEXADECIMAL** { **ON** | **OFF** }
| **ORDER** { **ON** | **OFF** }
| **PCTFREE** *percent-free-space*
| **QUOTE** *string*
| **QUOTES** { **ON** | **OFF** }
| **ROW DELIMITED BY** *string*
| **SKIP** *integer*
| **STRIP** { **ON** | **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*

パラメーター

column-name この句は、データのロード先となる 1 つ以上のカラムを指定するときに使用します。DEFAULTS が OFF に設定されている場合、カラムリストにないカラムは NULL になります。DEFAULTS オプションが ON で、カラムにデフォルト値が入っている場合は、その値が使用されます。DEFAULTS オプションが OFF で、NULL 入力不可のカラムがカラムリストから省かれている場合は、データベースサーバーは、空の文字列をカラムの型に変換しようとします。

カラムリストが指定されていると、ファイルに存在すると思われるカラムと、想定されるファイル内でのカラムの順序がカラムリストにリストされます。カラム名を繰り返すことはできません。リストにないカラム名は、NULL、0、空、または DEFAULT に設定されます。設定値は、カラムに NULL が許可されているかどうか、どのデータ型か、DEFAULTS が ON か OFF かによって異なります。LOAD TABLE によって無視される入力ファイル内のカラムは、カラム名 **filler()** を使用して指定できます。

load-source この句は、データのロード元となるデータソースを指定するときに使用します。さまざまなデータソースから、データをロードできます。次のリストは、サポートされているロードソースを示します。

FROM 句 この句は、ファイルを指定するときに使用します。*filename-expression* は文字列としてデータベースサーバーに渡されます。したがって、文字列は他の SQL 文字列と同じデータベースのフォーマット要件に従います。特に、次の点に注意してください。

○ディレクトリパスを示すには、円記号 (¥) を 2 つの円記号で表してください。ファイル *c:¥temp¥input.dat* から Employees テーブルにデータをロードする文は、次のようになります。

```
LOAD TABLE Employees
FROM 'c:¥¥temp¥¥input.dat' ...
```

○パス名はデータベースサーバーを基準にした相対パスを指定します。クライアントアプリケーションではありません。

○UNC パス名を使用すると、データベースサーバー以外のコンピューター上のファイルからデータをロードできます。

USING FILE 句 この句は、ファイルからデータをロードするときに使用します。これは、FROM *filename* 句を指定する場合と同義です。

LOAD TABLE 文を USING FILE 句とともに使用する場合、進行状況メッセージを要求できます。

また、Progress 接続プロパティを使用して、文がどの程度実行されたかを確認することもできます。

USING CLIENT FILE 句 この句は、クライアントコンピューター上のファイルからデータをロードするときに使用します。データベースサーバーが *client-filename-expression* からデータを取り出す場合、データはサーバーのメモリに実体化されないため、ファイルにはデータベースサーバーでの BLOB 式サイズの制限が適用されません。したがって、クライアントファイルには任意のサイズを指定できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

テーブルがクライアントファイルから読み込まれている場合、ファイル名のロギングはできません。ロギングタイプが指定されていない場合、WITH CONTENT LOGGING が使用されます。

LOAD TABLE 文を USING CLIENT FILE 句とともに使用する場合、進行状況メッセージを要求できます。

また、Progress 接続プロパティを使用して、文がどの程度実行されたかを確認することもできます。

USING VALUE 句　この句は、CHAR 型、NCHAR 型、BINARY 型、または LONG BINARY 型の式や、BLOB 文字列からデータをロードするときに使用します。次は、この句を使用する方法の例です。

○次の構文は、xp_read_file システムプロシージャーを使用して、対象ファイルからロードする値を取得します。

```
... USING VALUE xp_read_file( 'filename' )...
```

○次の構文は、値を直接指定して 2 つのローを挿入します。値はそれぞれ 4 と 5 です。

```
... USING VALUE '4¥n5'...
```

○次の構文は、READ_CLIENT_FILE 関数の結果を値として使用します。

```
... USING VALUE READ_CLIENT_FILE( client-filename-expression )
```

この場合、セマンティック上同じであるため、USING CLIENT FILE client-filename-expression も指定できます。

LOAD TABLE 文で ENCODING 句を指定しない場合、*value-expression* が CHAR 型または BINARY 型のときはデータベース文字セット (db_charset) が、*value-expression* が NCHAR 型のときは NCHAR データベース文字セット (nchar_charset) が *value-expression* のエンコードと見なされます。

USING COLUMN 句　この句は、別のテーブルの単一カラムからデータをロードするときに使用します。データベースサーバーは、リカバリ時に LOAD TABLE...WITH CONTENT LOGGING 文をリプレイしてトランザクションログをリプレイする場合に、この句を使用します。LOAD TABLE...WITH CONTENT LOGGING 文のトランザクションログレコードは、元の入力ファイルのまとまりで構成されます。リカバリ時にデータベースサーバーがトランザクションログ内でこうしたまとまりを見つけた場合は、このまとまりがテンポラリテーブルにロードされてから、元のロード操作のデータがすべてロードされます。

USING COLUMN 句では次の句が必要です。

○ ***table-name***　データのロード元となるカラムを含むベーステーブルまたはテンポラリテーブルの名前。リカバリ時にデータベースサーバーがトランザクションログから使用する場合、解析されてロードされるローのまとまりを保持するのはこのテーブルです。

○ ***column-name***　ロードするローのまとまりを保持する *table-name* 内のカラム名。

○ ***column-list***　送信先テーブルの 1 つまたは複数のカラムであり、データのロード前にローをソートするために使用。*column-list* は、プライマリキーやカラムリスト内に含まれる NULL

入力不可のカラムのユニークインデックスなど、検証可能な一意の値セットにする必要があります。

***load-option* 句**　さまざまなロードオプションを使用して、データのロード方法を制御できます。次のリストは、サポートされているロードオプションを示します。

○ **BYTE ORDER MARK 句**　この句は、データの最初でサーバーがバイトオーダーマーク (BOM) を検索して解釈するかどうかを指定するときに使用します。デフォルトでは、このオプションは ON になります。BYTE ORDER MARK が OFF の場合、サーバーは BOM を検索しません。

ENCODING 句が指定されている場合

- BYTE ORDER MARK オプションが ON のときに、UTF-16BE または UTF-16LE などのエンディアンを持つ UTF-16 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの検証に使用されます。間違ったエンディアンを指定すると、エラーが返されます。
- BYTE ORDER MARK オプションが ON のときに、明示的なエンディアンのない UTF-16 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの確認に使用されます。BOM がない場合は、オペレーティングシステムのエンディアンであると想定されます。
- BYTE ORDER MARK オプションが ON のときに UTF-8 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は無視されます。

ENCODING 句が指定されていない場合

- ENCODING 句を指定しないで BYTE ORDER MARK オプションを ON にすると、サーバーは入力データの最初で BOM を検索します。BOM が見つかると、BOM のエンコード (UTF-16BE、UTF-16LE、または UTF-8) に基づいてソースエンコードが自動的に選択され、BOM はロードするデータの一部であるとは見なされなくなります。BOM が見つからなかった場合、データベースの CHAR エンコードが使用されます。
- ENCODING 句を指定せず、BYTE ORDER MARK オプションが OFF である場合、データベースの CHAR エンコーディングが使用され、データベースサーバーはデータの最初で BOM を検索したり、解釈したりしません。

○ **CHECK CONSTRAINTS 句**　この句は、ロード時に制約をチェックするかどうかを制御するときに使用します。CHECK CONSTRAINTS はデフォルトで ON ですが、アンロードユーティリティ (dbunload) は CHECK CONSTRAINTS を OFF に設定して LOAD TABLE 文を書き出します。CHECK CONSTRAINTS を OFF に設定すると、データベースの再構築などの場合に役立つ検査制約が無効になります。まだ作成されていないユーザー定義関数を呼び出す検査制約がテーブルにあると、CHECK CONSTRAINTS が OFF に設定されていない場合、再構築は失敗します。

○ **COMMENTS INTRODUCED BY 句**　この句は、データファイルに使用する文字列を指定して、コメントを導入するときに使用します。使用すると、LOAD TABLE は文字列 *comment-prefix* で始まる行を無視します。たとえば、次の文で、// で始まる *input.dat* の行は無視されます。

```
LOAD TABLE Employees FROM 'c:¥¥temp¥¥input.dat' COMMENTS INTRODUCED BY '//' ...
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

コメントは新しい行の先頭にのみ指定できます。

COMMENTS INTRODUCED BY を省略する場合、コメントがデータとして解釈されるため、データファイルにはコメントを含めないでください。

○ **COMPRESSED 句**　ロードするデータが入力ファイル内で圧縮されている場合に COMPRESSED を指定します。データベースサーバーはデータを解凍してからロードします。データが圧縮されていないときに COMPRESSED を指定した場合は、LOAD が失敗してエラーが返されます。

データベースサーバーで入力ファイル内のデータが圧縮されているかどうかを判断できるようにするには、AUTO COMPRESSED を指定します。圧縮されていた場合、データベースサーバーはデータを解凍してからロードします。

入力ファイル内のデータが圧縮されていないことを指定するには、NOT COMPRESSED を指定します。データが圧縮されているときにデータベースサーバーでそのデータを解凍しない場合にも、NOT COMPRESSED を指定できます。この場合、データは圧縮された状態でデータベースに残ります。ただし、ファイルが暗号化されて圧縮されている場合、NOT COMPRESSED なしで NOT ENCRYPTED を使用することはできません。

○ **COMPUTES 句**　デフォルトでは、このオプションは ON です。この場合、計算カラムの再計算が有効になります。COMPUTES を OFF に設定すると、計算カラムの再計算が無効になります。COMPUTES OFF は、たとえば、データベースを再構築するとき、まだ作成されていないユーザー定義関数を呼び出す計算カラムがテーブルの中にある場合に役立ちます。この場合は、COMPUTES を OFF に設定しないと再構築に失敗します。

アンロードユーティリティ (dbunload) は、COMPUTES を OFF に設定して LOAD TABLE 文を書き出します。

○ **DEFAULTS 句**　デフォルトでは、DEFAULTS は OFF に設定されています。DEFAULTS が OFF の場合は、カラムリストにないカラムすべてに NULL が割り当てられます。DEFAULTS が OFF に設定されており、NULL 入力不可のカラムがカラムリストから省かれている場合は、データベースサーバーは、空の文字列をカラムの型に変換しようとします。DEFAULTS が ON に設定されており、カラムにデフォルト値が入っている場合は、その値が使用されます。

○ **DELIMITED BY 句**　この句は、カラムデリミター文字を指定するときに使用します。デフォルトのカラムデリミター文字列はカンマです。ただし、最長で 255 バイトの文字列を指定できます。たとえば、... DELIMITED BY '###' ... などです。指定するデリミターは文字列を用い、引用符で囲む必要があります。タブで区切った値を指定する場合、タブ文字を表す 16 進のエスケープシーケンス (9) を使用して、... DELIMITED BY '¥x09' ... のように指定します。

○ **ENCODING 句**　この句は、データベースにロードするデータに使用する文字エンコードを指定するときに使用します。ENCODING 句は、BCP フォーマットでは使用できません。

変換エラーがロード操作時に発生した場合、on_charset_conversion_failure オプションの設定に基づいてレポートされます。

データ内のバイトオーダーマークを解釈するには、BYTE ORDER 句を指定します。

ENCODING 句が指定されている場合

- BYTE ORDER MARK オプションが ON のときに、UTF-16BE または UTF-16LE などのエンディアンを持つ UTF-16 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの検証に使用されます。間違ったエンディアンを指定すると、エラーが返されます。
- BYTE ORDER MARK オプションが ON のときに、明示的なエンディアンのない UTF-16 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの確認に使用されます。BOM がない場合は、オペレーティングシステムのエンディアンであると想定されます。
- BYTE ORDER MARK オプションが ON のときに UTF-8 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は無視されます。

ENCODING 句が指定されていない場合

- ENCODING 句を指定しないで BYTE ORDER MARK オプションを ON にすると、サーバーは入力データの最初で BOM を検索します。BOM が見つかると、BOM のエンコード (UTF-16BE、UTF-16LE、または UTF-8) に基づいてソースエンコードが自動的に選択され、BOM はロードするデータの一部であるとは見なされなくなります。
- ENCODING 句を指定せず、BYTE ORDER MARK オプションが OFF である場合、データベースの CHAR エンコーディングが使用され、データベースサーバーはデータの最初で BOM を検索したり、解釈したりしません。

○ **ENCRYPTED 句**　この句は、暗号化設定を指定するときに使用します。暗号化されたデータをロードする場合は、ENCRYPTED KEY に続けて、入力ファイル内のデータ暗号化に使用するキーを指定します。

入力ファイル内のデータが暗号化されていないことを指定するには、NOT ENCRYPTED を指定します。データが暗号化されているときにデータベースサーバーでそのデータを復号しない場合にも、NOT ENCRYPTED を指定できます。この場合、データは暗号化された状態でデータベースに残ります。ただし、ファイルが暗号化されて圧縮されている場合、NOT COMPRESSED なしで NOT ENCRYPTED を使用することはできません。

○ **ESCAPE CHARACTER 句**　この句は、データで使用するエスケープ文字を指定するときに使用します。16 進のコードと記号として格納されている文字に使用するデフォルトのエスケープ文字は円記号 (¥) です。たとえば、¥x0A は改行文字です。エスケープ文字は、ESCAPE CHARACTER 句を使って変更することができます。たとえば、感嘆符 (!) をエスケープ文字として使用するには、次のように入力します。

```
ESCAPE CHARACTER '!'
```

エスケープ文字に指定する文字列は 1 マルチバイト文字を超えないようにすることをおすすめします。

○ **ESCAPES 句**　この句は、エスケープ文字を認識するかどうかを制御するときに使用します。ESCAPES を ON (デフォルト) にすると、データベースサーバーによってエスケープ文字 (デフォルトでは ¥) に続く文字が認識され、特殊文字として解釈されます。改行文字は ¥n という組み合わせとしてインクルードされ、他の文字はタブ文字の ¥x09 のような 16 進の ASCII のコードとしてデータにインクルードされます。2 つの円記号 (¥) は 1 つの円記号として解

釈されます。円記号 (¥) の後に n、x、X、¥ 以外の文字がある場合、それらは別々の文字と解釈されます。たとえば、¥q であれば、円記号と q が挿入されます。エスケープ文字に指定する文字列は 1 マルチバイト文字を超えないようにすることをおすすめします。

○ **FORMAT TEXT**　FORMAT TEXT (デフォルト) を選択すると、(ENCODING オプションで定義したとおりに) 入力行は文字と見なされ、1 行あたり 1 つのローで構成され、カラムデリミター文字列によって値が区切られます。

○ **FORMAT BCP**　FORMAT BCP を指定すると、Adaptive Server Enterprise 生成の BCP アウトファイルを読み込むことができます。

○ **FORMAT SHAPEFILE**　FORMAT SHAPEFILE を指定して、ESRI シェイプファイルをロードします。シェイプファイルは、データベースサーバーコンピューター上にある必要があり、FROM *filename-expression* または USING FILE *filename-expression* を使用してロードする必要があります。ここで、*filename-expression* は、ファイル拡張子が *.shp* の ESRI シェイプファイルを示します。関連付けられている *.shx* ファイルと *.dbf* ファイルは、*.shp* ファイルと同じディレクトリに配置し、ベースファイル名を同じ名前にしてください。

FORMAT SHAPEFILE については、ENCODING 句を指定しないと、エンコードはデフォルトで ISO-8859-1 になります。

FORMAT SHAPEFILE を指定する場合は、次のロードオプションを使用できます。

- CHECK CONSTRAINTS
- COMPUTES
- DEFAULTS
- ENCODING
- ORDER
- PCTFREE
- WITH CHECKPOINT
- WITH ....LOGGING

LOAD TABLE 文はロードしている 2 番目のカラムタイプから SRID を取得します。たとえば、2 番目のカラムをタイプ ST_Geometry(SRID=4326) で作成した場合、ジオメトリは SRID4326 を使用してロードされます。2 番目のカラムのタイプが ST_Geometry (明示的な SRID でない) の場合、ジオメトリは SRID 0 を使用してロードされます。

○ **FORMAT XML**　FORMAT XML を指定する場合は、次のロードオプションを使用できます。

- BYTE ORDER MARK
- CHECK CONSTRAINTS
- COMPRESSED
- COMPUTES
- DEFAULTS
- ENCODING
- ENCRYPTED
- ORDER
- PCTFREE
- WITH CHECKPOINT
- WITH ....LOGGING

FORMAT XML を使用する場合、入力ファイルは openxml システムプロシージャーを使用するクエリと同じように解析されます。SQL 文の引数は、システムプロシージャーのパラメーターと次のように対応します。

LOAD TABLE 文の句	openxml システムプロシージャー引数	詳細
row-xpath	*xpath*	
–	*flags*	openxml の *flags* 引数に対応する FORMAT XML で値を指定する方法はありません。
NAMESPACES	*namespaces*	

FORMAT XML 句では、次のパラメーターが使用されます。

- ***row-xpath*** XPath クエリを含む文字列または変数。XPath を使用すると、問い合わせる XML ドキュメントの構造を記述するパターンを指定できます。この引数にある XPath パターンによって、XML ドキュメントからノードが選択されます。*row-xpath* 引数の XPath クエリに一致する各ノードが、テーブルにローを 1 つずつ生成します。

  FORMAT XML 句の *row-xpath* 引数に指定できるのは、メタプロパティのみです。メタプロパティは、属性のように XPath クエリ内でアクセスされます。*namespaces* が指定されていない場合、デフォルトでプレフィクス mp が Uniform Resource Identifier (URI) urn:ianywhere-com:sa-xpath-metaprop にバインドされます。*namespace* が指定された場合、この URI は、クエリのメタプロパティにアクセスするために mp または他のプレフィクスにバインドされます。メタプロパティ名の大文字と小文字は区別されます。次のメタプロパティがサポートされています。

  - **@mp:id** XML ドキュメント内のユニークなノードの ID を返します。データベースサーバーが再起動されると、ドキュメント内の指定されたノードの ID が変更される場合もあります。このメタプロパティの値は、ドキュメントの順序で増えていきます。

  - **@mp:localname** ノードの名前のローカル部分を返します。ノードに名前がない場合は NULL を返します。

  - **@mp:prefix** ノードの名前のプレフィクス部分を返します。ノードに名前がない場合または名前にプレフィクスが付いていない場合は NULL を返します。

  - **@mp:namespaceuri** ノードが含まれているネームスペースの URI を返します。ノードがネームスペースに含まれていない場合は NULL を返します。

  - **@mp:xmltext** XML ドキュメントのサブツリーを XML 形式で返します。たとえば、内部ノードを照合する場合、このメタプロパティを使用して、下位のテキストノードの連結値ではなく、XML 文字列を返します。

- ***column-xpath*** 結果セットのスキーマと、結果セット内で各カラムに値がどのように格納されるかを指定する文字列または変数。FORMAT XML 句の式が複数のノードと一致

した場合、ドキュメントの順序における最初のノードのみが使用されます。ノードがテキストノードではない場合、テキストノードの下位ノードをすべて追加することにより結果を検索します。FORMAT XML 句の式がどのノードにも一致しない場合は、そのローのカラムは NULL になります。

- ***namespace*** XML ドキュメントを含む文字列または変数。クエリのスコープ内のネームスペースは、ドキュメントのルート要素から取得されます。

○ **HEXADECIMAL 句** この句は、バイナリ値を 16 進として読み込むかどうかを指定するときに使用します。デフォルトでは、HEXADECIMAL は ON です。HEXADECIMAL ON の場合、バイナリカラム値は **0x***nnnnnn*... として読み込まれます。ここで、0x は 1 つのゼロの後に 1 つの x が続きます。*n* はそれぞれ 16 進数の数字です。マルチバイト文字セットを扱う場合は、HEXADECIMAL ON を使用することが重要です。

HEXADECIMAL 句は、FORMAT TEXT 句を指定した場合にのみ使用できます。

○ **ORDER 句** この句は、ロード時にデータをソートするかどうかを指定するときに使用します。ORDER のデフォルトは ON です。ORDER が ON に設定され、クラスタードインデックスが宣言されている場合、LOAD TABLE はクラスタードインデックスに従って入力データをソートし、同じ順序でローを挿入します。ロードするデータがすでにソート済みの場合は、ORDER を OFF に設定してください。

○ **PCTFREE 句** この句は、各テーブルページに確保する空き領域の割合を指定するときに使用します。この設定は、テーブルに対する永続的な設定を上書きしますが、それはロード中のみ実行されます。また、ロードされているデータだけが対象となります。*percent-free-space* の値は、0 ～ 100 の間の整数です。0 を指定すると、各ページに空き領域を残さず、各ページを完全にパックします。高い値に設定すると、各ローは単独でページに挿入されます。

○ **QUOTE 句** QUOTE 句は TEXT データ専用です。*string* は文字列値を囲みます。デフォルトは一重引用符 (アポストロフィ) です。

○ **QUOTES 句** この句は、文字列を引用符で囲むかどうかを指定するときに使用します。QUOTES を ON (デフォルト) に設定すると、LOAD TABLE 文は引用符文字で囲まれた文字列を探します。QUOTES 句が省略されている場合、引用符はアポストロフィ (一重引用符) または引用符 (二重引用符) のいずれかになり、文字列に最初に登場するこれらの文字がその文字列の引用符として扱われます。文字列の終わりには先頭にあるものと同じ引用符が必要です。

QUOTES を ON に設定すると、カラムデリミター文字列をカラム値の中に入れることができます。また、引用符文字は値の一部とは見なされません。したがって、次の行はアドレスにカンマがあるかどうかに関係なく、3 つではなく 2 つの値として扱われます。また、アドレスを囲む引用符は、データベースへは挿入されません。

```
'123 High Street, Anytown',(715)398-2354
```

値の中に引用符文字を含めるには、QUOTES を ON に設定して、2 つの引用符を使用します。次の行は、第 3 カラムの中へ一重引用符文字である値を入れます。

```
'123 High Street, Anytown','(715)398-2354',""
```

○ **ROW DELIMITED BY 句** この句は、入力レコードの末尾を示す文字列を指定するときに使用します。デフォルトのデリミター文字列は改行 (¥n) です。ただし、最長で 255 バイトの文

字列を指定できます。たとえば、ROW DELIMITED BY '###' などです。タブで区切った値を指定する場合、タブ文字を表す 16 進のエスケープシーケンス (9) を使用して、ROW DELIMITED BY '¥x09' のように指定します。デリミター文字列に ¥n が含まれる場合、¥r¥n または ¥n のどちらかに一致します。

○ **SKIP 句**　この句は、ファイルの最初の数行を無視するかどうかを指定するときに使用します。*integer* 引数では、スキップする行数を指定します。たとえば、この句を使用して、カラム見出しを含む行をスキップできます。ローデリミターがデフォルト (改行) でない場合、引用符付きの文字列に埋め込まれたローデリミターがデータに含まれると、スキップが正しく動作しないことがあります。

○ **STRIP 句**　この句は、引用符で囲まれていない値を挿入する前に、この値の先行ブランクまたは後続ブランクを削除するかどうかを指定するときに使用します。STRIP オプションは次のオプションを受け入れます。

- **STRIP OFF**　先行ブランクも後続ブランクも削除しません。
- **STRIP LTRIM**　先行ブランクを削除します。
- **STRIP RTRIM**　後続ブランクを削除します。
- **STRIP BOTH**　先行ブランクと後続ブランクの両方を削除します。
- **STRIP ON**　廃止予定。STRIP RTRIM と同等です。

STRIP 動作は QUOTES 句と結び付いています。QUOTES OFF を指定すると、STRIP OFF、STRIP LTRIM、STRIP RTRIM、STRIP BOTH は正確に文字どおりに機能します。QUOTES 句を指定しなかったり、QUOTES ON を指定したりすると、引用符の付いていない文字列は常に左トリムおよび右トリムされます (ただし、文字列が右トリムされないようにする場合は、STRIP OFF または STRIP LTRIM を指定します)。

○ **WITH CHECKPOINT 句**　この句は、チェックポイントを実行するかどうかを指定するときに使用します。デフォルト設定は OFF です。この句を ON に設定すると、文が正常に完了し、ロギングされた後にチェックポイントが発行されます。この句が ON に設定されているとき、チェックポイントを発行する前にデータベースの自動リカバリが必要な場合で、FILE NAME LOGGING を使用している場合には、リカバリを正しく完了するために、テーブルのロードに使用されるデータファイルが存在している必要があります。WITH CHECKPOINT ON を指定し、リカバリが続いて必要な場合、リカバリはチェックポイントの後で始まり、データファイルは存在する必要はありません。

データベースが破損したため、バックアップを使用して現在のログファイルを適用する必要がある場合は、FILE NAME LOGGING を使用していれば、この句に何が指定されているかに関係なく、データファイルが必要となります。

○ **WITH { FILE NAME | ROW | CONTENT } LOGGING**　この句は、ロード操作時にトランザクションログに記録される詳細レベルを制御するときに使用します。ロギングのレベルを次に示します。

- **WITH FILE NAME LOGGING 句**　WITH FILE NAME LOGGING 句を指定すると、LOAD TABLE 文のみがトランザクションログに記録されます。リカバリ時にトランザクション

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ログを使用する場合に結果が矛盾しないようにするには、元のロード操作に使用したファイルが元のロケーションにあり、そのファイルに元のデータが含まれている必要があります。このレベルのロギングが、最もパフォーマンスが高いです。ただし、データベースがミラーリングまたは同期を行っている場合は、このレベルを使用しないでください。また、式またはクライアントファイルからロードする場合にも、このレベルは使用できません。

LOAD TABLE 文でロギングレベルを指定しない場合、次を指定したときは WITH ROW LOGGING がデフォルトのレベルになります。

○ FROM *filename-expression*
○ USING FILE *filename-expression*

● **WITH ROW LOGGING 句** WITH ROW LOGGING 句を指定すると、各ローのロードが INSERT 文としてトランザクションログに記録されます。FROM *filename-expression* または USING FILE *filename-expression* を使用している場合、同期を行っているデータベースやデータベースミラーリングのデフォルトになっているデータベースには、このレベルのロギングをおすすめします。ただし、大量のデータをロードする場合、このロギングタイプがパフォーマンスに影響したり、トランザクションログがかなり長くなったりする可能性があります。

非決定的な値がない場合は、WITH CONTENT LOGGING が最もパフォーマンスが高くなる可能性があります。

また、このレベルは、計算カラムや CURRENT TIMESTAMP のデフォルトなどの非決定的な値がロード先のテーブルに含まれている場合に最適です。

● **WITH CONTENT LOGGING 句** WITH CONTENT LOGGING 句を指定すると、データベースサーバーは入力ファイルをまとまりとしてトランザクションログにコピーします。これらのまとまりは、リカバリ時にトランザクションログからなど、後から入力ファイルのコピーに再構成できます。大量のデータをロードする場合、このロギングタイプはパフォーマンスへの影響が非常に少なく、強力にデータを保護できますが、トランザクションログが長くなります。このロギングレベルは、ミラーリングを行っているデータベース、または非決定的な値がないときに後でリカバリするために元のデータファイルを維持しない方が望ましい場合におすすめします。

データベースが同期を行っている場合は、WITH CONTENT LOGGING 句を使用できません。WITH CONTENT LOGGING 句はテーブルがクライアントファイルからロードされるときに必要です。

LOAD TABLE 文でロギングレベルを指定しない場合、次を指定したときは WITH CONTENT LOGGING がデフォルトのレベルになります。

○USING CLIENT FILE *client-filename-expression*

○USING VALUE *value-expression*

○USING COLUMN *column-expression*

statistics-limitation-option LOAD TABLE の実行中に統計が生成されるカラムを制限できます。制限しないと、すべてのカラムに対して統計が生成されます。統計が一部のカラムで使用さ

れないことが確かな場合のみ、この句を使用してください。ON ALL COLUMNS (デフォルト)、OFF、ON KEY COLUMNS、または統計を生成するカラムのリストを指定できます。

備考

LOAD TABLE を使うと、ファイルからデータベーステーブルの中へ効率よく大量の挿入を行えます。LOAD TABLE は、Interactive SQL 文の INPUT より効率的です。

LOAD TABLE は、テーブル全体に書き込みロックをかけます。ベーステーブル、グローバルテンポラリテーブル、ローカルテンポラリテーブルの場合、コミットが実行されます。

即時テキストインデックスが構築されているテーブル、または即時ビューから参照されているテーブルで LOAD TABLE を使用しようとすると、ロードは失敗します。即時テキストインデックス以外のインデックスまたはマテリアライズドビューでこのようなことは発生しません。ただし、従属するインデックスやマテリアライズドビューのデータをトランケートしてから LOAD TABLE 文を実行し、その後、インデックスとビューをリフレッシュすることを強くおすすめします。

作成時に ON COMMIT DELETE ROWS が明示的またはデフォルトで指定されている場合、テンポラリテーブルには LOAD TABLE 文を使用しないでください。ただし、ON COMMIT PRESERVE ROWS または NOT TRANSACTIONAL が指定されている場合は、LOAD TABLE を**使用できます**。

FORMAT TEXT の場合、値を指定しないことが、NULL 値を指定することになります。たとえば、3 つの値が予期されている場合に、ファイルに 1,,'Fred', が含まれていると、挿入される値は 1、NULL、Fred になります。ファイルに 1,2, が含まれる場合、値 1、2、NULL が挿入されます。空白のみで構成される値も、NULL 値と見なされます。ファイルに 1, ,'Fred', が含まれる場合、値 1、NULL、Fred が挿入されます。他の値はすべて NULL 以外と見なされます。たとえば、'' (一重引用符が 2 つ続く) は空の文字列です。'NULL' は 4 文字の文字列です。

LOAD TABLE でロードされるカラムが NULL 値を許容しておらず、ファイル値が NULL の場合、数値カラムには値 0 (ゼロ)、文字カラムには空の文字列 ('') が指定されます。LOAD TABLE でロードされるカラムが NULL 値を許容し、ファイル値が NULL の場合、カラム値は (すべての型で) NULL になります。

テーブルにカラム a、b、c、入力データに a、b、c があるとき、データのロード先カラムとして a と b のみを LOAD 文に指定すると、次の値がカラム c に挿入されます。

- DEFAULTS ON が指定されると、カラム c にデフォルト値が入っている場合は、デフォルト値が使用されます。
- カラム c が NULL 値を許容するときにデフォルトが定義されていない場合は、NULL が使用されます。
- カラム c が NULL を許容しないときにデフォルトが定義されていない場合は、カラムのデータ型に応じて、ゼロ (0) または空の文字列 ('') が使用されるか、またはエラーが返されます。
- **LOAD TABLE とデータベースミラーリング**　データベースミラーリングを使用しているときにベーステーブルに対して LOAD TABLE 文を実行する場合は、文のロギングレベルとして、WITH ROW LOGGING または WITH CONTENT LOGGING を指定する必要があります。これ

らの句を指定することで、ロードされたデータをトランザクションログに記録して、ミラーリングデータベースにもロードできるようになります。これらの句が指定されていない場合は、エラーがレポートされます。

- **LOAD TABLE とカラム統計**　LOAD TABLE は、テーブルのカラムに関するヒストグラムを作成するために、データのロード時にカラム統計を取得します。ヒストグラムは、オプティマイザーによって使用されます。

  次に、ロードとカラム統計に関する追加のヒントを示します。

  ○LOAD TABLE は、今後の使用のためにベーステーブルに関する統計情報を保存します。グローバルテンポラリテーブルに関する統計情報は保存しません。

  ○以前にデータが含まれていた可能性のある空のテーブルにデータをロードする場合は、LOAD TABLE 文を実行する前にカラムの統計を削除します。

  ○カラムで LOAD TABLE が実行されたときにカラムの統計が存在すると、このカラムの統計は再計算**されません**。代わりに、既存の統計に新しいデータの統計が挿入されます。これは、既存のカラム統計が古い情報である場合、カラムに新しいデータをロードした後も依然として古いデータのまま残ることを意味します。カラム統計が古いことが考えられる場合は、LOAD TABLE 文の実行の前または後に更新することを検討します。

  ○LOAD TABLE は、テーブルに 5 つ以上のローがある場合にのみ統計情報を追加します。テーブルのローの数が 5 つ以上の場合、ヒストグラムは次のように変更されます。

テーブル内の既存データの有無	ヒストグラムの有無	実行されるアクション
あり	あり	既存のヒストグラムに変更を統合する
あり	なし	ヒストグラムを構築しない
なし	あり	既存のヒストグラムに変更を統合する
なし	なし	新しいヒストグラムを作成する

  ○LOAD TABLE は、ロードされたローの 90% を超える値が NULL 値の場合、カラムの統計情報を生成しません。

- **動的に構成されたファイル名の使用**　変数にファイル名を割り当て、その変数名を LOAD TABLE 文で使用することによって、動的に構成されたファイル名を使用して、LOAD TABLE 文を実行できます。

パーミッション

LOAD TABLE 文を実行するのに必要なパーミッションは、データベースサーバーの -gl オプションによって次のように異なります。

- -gl オプションが ALL に設定されている場合、テーブルの所有者であること、DBA 権限を持っていること、または ALTER 権限を持っていることのいずれかの条件に該当している必要があります。
- -gl オプションが DBA に設定されている場合、DBA 権限を持っていることが必要です。
- -gl オプションが NONE に設定されている場合、LOAD TABLE は使用できません。

テーブルに排他ロックを実行してください。

クライアントコンピューターにあるファイルからデータを読み込む場合

- READCLIENTFILE 権限が必要です。
- 読み込み元のディレクトリに対する読み込みパーミッションが必要です。
- allow_read_client_file データベースオプションが有効になっている必要があります。
- read_client_file セキュア機能が有効になっている必要があります。

関連する動作

オートコミット。

挿入は、WITH ROW LOGGING 句が指定された場合を除き、ログファイルに記録されません。そのため、ロギングタイプによっては、エラーが発生した場合に、挿入されたローを回復できないことがあります。ローを回復する必要があり、WITH FILE NAME LOGGING を使用している場合は、元のファイルが必要です。また、Mobile Link クライアントとして使用されるデータベースまたは SQL Remote レプリケーションに関連するデータベースでは、WITH ROW LOGGING 句のない LOAD TABLE 文は使用しないでください。これは、これらのテクノロジでは、ログファイルを解析して変更がレプリケートされるためです。

LOAD TABLE 文は、テーブルに関連したトリガーは起動しません。

チェックポイントは操作開始時に実行されます。WITH CHECKPOINT ON を指定すると、2 番目のチェックポイントが終了時に実行されます。

データを大量にロードすると、カラム統計が更新されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「クラスタードインデックス」『SQL Anywhere サーバー SQL の使用法』
- 「サポートされている文字セット」『SQL Anywhere サーバー データベース管理』
- 「TRUNCATE 文」961 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「on_charset_conversion_failure オプション」『SQL Anywhere サーバー データベース管理』
- Progress 接続プロパティ『SQL Anywhere サーバー データベース管理』
- 「progress_messages オプション」『SQL Anywhere サーバー データベース管理』
- 「データインポート」『SQL Anywhere サーバー SQL の使用法』
- 「LOAD TABLE 文」814 ページ
- 「UNLOAD 文」966 ページ
- 「INSERT 文」802 ページ
- 「INPUT 文 [Interactive SQL]」794 ページ
- 「クライアントコンピューター上のデータへのアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「オプティマイザーの推定と統計」『SQL Anywhere サーバー SQL の使用法』
- 「-sf dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「allow_read_client_file オプション」『SQL Anywhere サーバー データベース管理』
- 「READCLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』
- 「カラム統計の更新」『SQL Anywhere サーバー SQL の使用法』
- 「-gl dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「CREATE TABLE 文」651 ページ
- 「ESRI シェイプファイルのサポート」『SQL Anywhere サーバー 空間データサポート』
- 「データのインポートとエクスポート」『SQL Anywhere サーバー SQL の使用法』
- 「LOAD TABLE 文を使用したデータのインポート」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次に、LOAD TABLE の例を示します。まずテーブルを作成し、次に *input.txt* というファイルを使用してテーブルにデータをロードします。

```
CREATE TABLE t( a CHAR(100), let_me_default INT DEFAULT 1, c CHAR(100) );
```

次に、ファイル *input.txt* の内容を示します。

```
ignore_me, this_is_for_column_c, this_is_for_column_a
```

次の LOAD 文は、*input.txt* というファイルをロードします。

```
LOAD TABLE T ( filler(), c, a ) FROM 'input.txt' FORMAT TEXT DEFAULTS ON;
```

コマンド SELECT * FROM t は、次の結果セットを返します。

a	let_me_default	c
this_is_for_column_a	1	this_is_for_column_c

次の例は、EXECUTE IMMEDIATE 文を使用して、動的に構成されるファイル名で LOAD TABLE 文を実行します。

```
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;
```

次の例は、UTF-8 エンコードのテーブルデータを mytable にロードします。

```
LOAD TABLE mytable FROM 'mytable_data_in_utf8.dat' ENCODING 'UTF-8';
```

LOCK FEATURE 文

他の同時接続でデータベースサーバーの機能が使用されないようにします。

構文

LOCK FEATURE *feature-name* { **ON** | **OFF** }

feature-name :
synchronization schema
| **all**

パラメーター

feature-name ロックまたはロック解除する機能の名前。接続でロックされているすべての機能をロック解除するには、all を指定します。

ON | OFF 他の接続で機能が使用されないようにするには、ON を指定します。接続で機能を使用できるようにするには、OFF を指定します。

備考

同じ接続に対して 1 つの機能を複数回ロックすることはできません。現在の接続によってロックされていない機能をロック解除する場合に、機能名として all を指定しないと、エラーが返されます。1 つの機能が 2 つ以上の接続によってロックされている場合に、その機能を他の接続で使用できるようにするには、事前にすべての接続によってその機能がロック解除される必要があります。接続によって作成された機能ロックは、その接続が切断されると、削除されます。機能ロックは、データベースサーバーが停止すると、削除されます。

同期スキーマ機能がロックされている場合、他の接続では次の文を実行できません。

- START SYNCHRONIZATION SCHEMA CHANGE
- CREATE SYNCHRONIZATION SUBSCRIPTION
- DROP SYNCHRONIZATION SUBSCRIPTION
- ALTER SYNCHRONIZATION SUBSCRIPTION
- ALTER PUBLICATION

パーミッション

DBA 権限

関連する動作

なし

参照

- 「LOCK TABLE 文」831 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、他の接続で同期スキーマ機能が使用されないようにします。

```
LOCK FEATURE 'synchronization schema' ON;
```

LOCK TABLE 文

同時に実行されている他のトランザクションがテーブルにアクセスしたり、テーブルを修正したりすることを防止します。

構文

LOCK TABLE *table-name*
[WITH HOLD]
IN { SHARE | EXCLUSIVE } MODE

パラメーター

table-name テーブルの名前。テーブルとして、ビューではなくベーステーブルを指定してください。テンポラリテーブルのデータは現在の接続に固有のローカルなものであるため、グローバルテンポラリテーブルまたはローカルテンポラリテーブルでは、ロックは効力を持ちません。

WITH HOLD 句 この句は、接続が終了するまでテーブルをロックするときに指定します。この句を指定しない場合は、現在のトランザクションがコミットまたはロールバックされた時点でロックは解放されます。

IN SHARE MODE 句 この句は、テーブルの共有テーブルロックを取得して、他のトランザクションはテーブルを修正できないようにするが、読み込みアクセスは許可するときに指定します。トランザクションがテーブルに共有ロックをかけると、他のトランザクションが修正中のローにロックをかけていない場合に、テーブルのデータを変更できます。IN SHARE MODE 句が選択されている場合、個々のローの読み込みロックは取得されません。

IN EXCLUSIVE MODE 句 この句は、テーブルの排他テーブルロックを取得して、他のトランザクションがそのテーブルにアクセスできないようにするときに指定します。他のトランザクションは、クエリ、更新を含め、テーブルに対するどのようなアクションも実行できません。LOCK TABLE...IN EXCLUSIVE MODE などの文を使用してテーブルを排他的にロックすると、デフォルトの動作ではテーブルのローロックを取得しません。subsume_row_locks オプションを Off に設定すると、この動作を無効にできます。

備考

LOCK TABLE 文によって、現在の独立性レベルに関係なく、同時実行性をテーブルレベルで直接制御できます。

トランザクションの独立性レベルは、現在のトランザクションが要求を実行するときに設定されるロックを制御しますが、LOCK TABLE 文では、もっと明示的にテーブル内のローのロックを制御できます。

ビューには LOCK TABLE 文を実行できません。ただし、ベーステーブルに LOCK TABLE 文を実行すると、共有スキーマロックが作成され、このロックによって従属ビューがロックされます。共有スキーマロックは、トランザクションがコミットされるか、またはロールバックされるまで継続します。

パーミッション

SHARE モードのテーブルをロックするためには、SELECT 権限が必要です。

EXCLUSIVE モードでテーブルをロックするためには、そのテーブルの所有者であるか、DBA 権限が必要です。

関連する動作

ロックされたテーブルへのアクセスが必要な他のトランザクションは、遅延またはブロックされます。

参照

- 「テーブルロック」『SQL Anywhere サーバー SQL の使用法』
- 「SELECT 文」895 ページ
- 「sa_locks システムプロシージャー」1111 ページ
- 「ロックの仕組み」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、現在のトランザクションの実行中は、他のトランザクションに対して Customers テーブルの修正を禁止します。

```
LOCK TABLE Customers
IN SHARE MODE;
```

LOOP 文

文リストの実行を繰り返します。

構文

[*statement-label* :]
[**WHILE** *search-condition*] **LOOP**

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

statement-list
END LOOP [*statement-label*]

備考

WHILE と LOOP 文は制御文です。これを使って、*search-condition* が TRUE と評価するかぎり、SQL 文のリストを繰り返し実行できます。LEAVE 文を使って、END LOOP の後に記述されている最初の文から実行を再開できます。

終了の *statement-label* を指定する場合は、開始の *statement-label* と一致させます。

パーミッション

なし

関連する動作

なし

参照

- 「FOR 文」753 ページ
- 「 CONTINUE 文」520 ページ
- 「WHILE 文 [T-SQL]」988 ページ

標準と互換性

- **SQL/2008**　LOOP/END LOOP 文は、オプションの SQL/2008 言語機能 P002、"Computational completeness" の一部です。SQL/2008 では、WHILE DO/END WHILE 文は、言語機能 P002 の一部でもある別個の文です。SQL Anywhere でサポートされている構文の組み合わせ WHILE *search-condition* LOOP はベンダー拡張です。

- **Transact-SQL**　LOOP は Transact-SQL ダイアレクトではサポートされていません。Transact-SQL ストアドプロシージャー内でのループは Transact-SQL WHILE 文で実行されます。

例

プロシージャー内の While ループの例

```
...
SET i = 1;
WHILE i <= 10 LOOP
  INSERT INTO Counters( number ) VALUES ( i );
  SET i = i + 1;
END LOOP;
...
```

プロシージャー内の指定されたラベルのループの例

```
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
```

MERGE 文

テーブル、ビュー、プロシージャーの結果をテーブルまたはビューにマージします。

構文

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**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

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 :17000 よりも大きい正の整数または変数

option-name :*identifier*

option-value :
hostvar (許可されたインジケーター)
| *string*
| *identifier*
| *number*

table-hints :「FROM 句」758 ページを参照してください。

search-condition :「探索条件」34 ページを参照してください。

set-clause-list :「SET 文」927 ページを参照してください。

パラメーター

INTO 句　この句は、MERGE 文のターゲットとなるオブジェクトを定義するときに使用します。*target-object* には、ベーステーブル、通常のビュー、または派生テーブルの名前を指定できます。マテリアライズドビューは指定できません。派生テーブルまたはビューは、更新可能なクエリブロックである必要があります。たとえば、ビュー定義または派生テーブル定義に UNION、INTERSECT、EXCEPT、または GROUP BY が含まれている場合、MERGE 文のターゲットとなるオブジェクトとしては使用できません。

target-object が派生テーブルの場合、オプションの *into-column-list* を使用すると、派生テーブルのカラムに代替名を指定できます。このように使用する場合は、*into-column-list* のサイズが派生テーブルのカラムリストと一致し、2 つのリストの順序も同じである必要があります。

target-object がベーステーブルまたはビューの場合、*into-column-list* を使用すると、MERGE 文が終わるまでテーブルのカラムまたはビューのカラムのサブセットを関連として指定できます。

データベースサーバーは *into-column-list* を使用して、次の項目を解析します。

○WHEN MATCHED 句の中に SET 句のない UPDATE

○WHEN NOT MATCHED 句の中に VALUES 句のない INSERT

○ON 句の中の PRIMARY KEY 探索条件

○USING 句の中の WITH AUTO NAME 句

into-column-list を指定しない場合、*into-column-list* には *target-object* のすべてのカラムが含まれていると見なされます。

USING 句　この句は、マージ元のデータソースを定義するときに使用します。*source-object* には、ベーステーブル (テーブルヒントを含む)、ビュー、マテリアライズドビュー、派生テーブル、またはプロシージャーを指定できます。*source-object* が派生テーブルの場合は、*using-column-list* を指定できます。*using-column-list* を指定しない場合は、*source-object* のすべてのカラムが使用されます。

WITH AUTO NAME 句　この句は、*target-object* の *into-column-list* カラム内のカラムに一致するカラム名を、サーバーがマージ操作で自動的に使用するように指定するときに使用します。次の例は同等であり、WITH AUTO NAME を指定すると、*into-column-list* 内のカラムの順序が、*source-object* 内のカラム名に一致するように変更されることを示します。

```
... 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 句　この句は、*source-object* 内のローを *target-object* 内のローに一致させるための条件を指定するときに使用します。

target-object のプライマリキー定義に基づいて *source-object* のローを一致させるには、ON PRIMARY KEY を指定します。*source-object* にプライマリキーは必要ありません。ただし、*target-object* にはプライマリキーが必要です。ON PRIMARY KEY を指定すると、次のようになります。

○*target-object* がベーステーブルでない場合、またはプライマリキーがない場合はエラーが返されます。

○*into-column-list* 内に 1 つ以上のプライマリキーカラムが含まれていない場合はエラーが返されます。

○*into-column-list* 内のすべてのプライマリキーカラムが *using-column-list* 内の対応するカラムと一致している場合、*into-column-list* と *using-column-list* のカラム数が異なる可能性があります。たとえば、*into-column-list* が (I1, I2, I3)、*using-column-list* が (U1, U2)、プライマリキーカラムが (I2, I3) の場合、*target-object* のプライマリキーカラム (I3) が *using-column-list* で一致しないため、エラーが返されます。

○プライマリキーの定義に関係なく、*into-column-list* のプライマリキーカラムと *using-column-list* の式の一致は、*into-column-list* 内のプライマリキーカラムの位置に左右されます。たとえば、*target-object* のプライマリキーが (B, C) と定義されており、*into-column-list* が (E, C, F, A, D, B) であると想定します。ON PRIMARY KEY を指定すると、*target-object* のカラム B は *into-column-list* の 6 番目の位置にあるため、*using-column-list* の 6 番目の要素と比較されます。同様に、*target-object* のカラム C は *using-column-list* の 2 番目の要素と比較されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ON PRIMARY KEY は、対応する ON 条件の構文上の省略形です。たとえば、*into-column-list* が (I1, I2, ..I*n*)、対応する一致した *using-column-list* が (U1, U2, ..U*m*) であると仮定します。また、*target-object* のプライマリキーカラムが I1, I2, I3 であり、プライマリキーカラムがすべて *into-column-list* に含まれているとします。この場合、*merge-search-condition* は、連結された "I1=U1 AND I2=U2 AND I3=U3" と定義されます。

WHEN MATCHED 句と WHEN NOT MATCHED 句　WHEN MATCHED 句と WHEN NOT MATCHED 句は、*source-object* のローが *target-object* のローに一致する場合、または一致しない場合に実行するアクションを定義するときに使用します。アクションは、THEN キーワードの後に指定します。追加の AND 句を指定すると、一致するローのサブセットまたは一致しないローのサブセットに実行するアクションを制御できます。

ON 句を使用すると、*source-object* のローを、一致するローと一致しないローに分類する方法が決まります。*target-object* 内の少なくとも 1 つのローについて ON 句が TRUE の場合、*source-object* 内のローは一致するローと見なされます。*target-object* 内のどのローも ON 句が TRUE でない場合、*source-object* のローは一致しないローと見なされます。複数の WHEN MATCHED 句と WHEN NOT MATCHED 句を使用すると、一致するローと一致しないローのセットが切断のサブセットに分割されます。各サブセットは WHEN 句によって処理されます。WHEN MATCHED 句と WHEN NOT MATCHED 句は、MERGE 文に記載されている順序で処理されます。

WHEN MATCHED 句または WHEN NOT MATCHED 句の AND 句で指定した探索条件によって、候補となるローを特定の句で処理するかどうかが決まります。AND 句なしで WHEN MATCHED 句または WHEN NOT MATCHED 句を指定すると、AND 句の探索条件は TRUE であると見なされます。あるローが複数句の AND 条件を満たす場合、そのローは MERGE 文で最初に記載されている句によって処理されます。

同じ *target-object* のローを WHEN MATCHED 句のいずれかが複数回処理した場合は、エラーが返されます。1 つの *target-object* のローは、*source-object* の 2 つの異なる入力ローに一致する場合、同じ WHEN MATCHED 句の同じサブセットに複数回割り当てられます。

次の例では、*target-object* の Products で ID 300 のローが *source-object* の SalesOrderItems の 111 のローに一致するため、エラーが返されます。すべての一致は、WHEN MATCHED THEN UPDATE 句に対応する同じサブセットに属しています。

```
MERGE INTO Products
  USING SalesOrderItems S
  ON S.ProductID = Products.ID
  WHEN MATCHED THEN UPDATE SET Products.Quantity = 20;
```

「WHEN MATCHED」: 一致するローの場合、*match-action* には次のいずれかのアクションを指定できます。

○ **DELETE**　DELETE は、*target-object* のローを削除するときに指定します。

○ **RAISERROR**　RAISERROR は、マージ操作を終了し、変更をロールバックしてエラーを返すときに指定します。RAISERROR を指定すると、データベースサーバーはデフォルトで SQLSTATE 23510 および SQLCODE -1254 を返します。

必要に応じて、RAISERROR キーワードの後に *error-number* パラメーターを指定し、返される SQLCODE をカスタマイズできます。カスタム SQLCODE には 17000 よりも大きい正の

整数を指定してください。数または変数のいずれとしても指定できます。カスタム SQLCODE を指定すると、負の数が返されます。

たとえば、WHEN MATCHED AND search-condition THEN RAISERROR 17001 を指定すると、WHEN 句の条件を満たすローが検出された場合、マージ操作は失敗し、変更がロールバックされ、SQLSTATE 23510 と SQLCODE -17001 を持つエラーが返されます。

○ **SKIP**　SKIP は、ローをスキップするときに指定します。何のアクションも行われません。

○ **UPDATE**　UPDATE SET は、*set-item* 値を使用してローを更新するときに指定します。*set-item* は単純な割り当て式であり、カラムは *expression* の値に設定されます。*expression* には制限がありません。また、DEFAULT を指定すると、カラムをカラム用に定義されているデフォルトに設定できます。

たとえば、UPDATE SET target-column1=DEFAULT, target-column2=source-column2 によって、target-column1 はデフォルト値に設定され、target-column2 は *source-object* の source-column2 の修正ローと同じに設定されます。

SET 句を指定しない場合、SET 句は *into-column-list* と *using-column-list* によって定義されます。たとえば、*into-column-list* が (I1, I2, ..I*n*)、*using-column-list* が (U1, U2, ..U*n*) である場合、SET 句には "SET I1=U1 , I2=U2 , .. In=Un" が使用されます。

「WHEN NOT MATCHED」：一致しないローの場合、*non-match-action* には次のいずれかのアクションを指定できます。

○ **INSERT**　INSERT...VALUES は、指定した値を使用してローを挿入するときに指定します。VALUES 句なしで INSERT 句を指定すると、VALUES 句は *into-column-list* と *using-column-list* によって定義されます。たとえば、*into-column-list* が (I1, I2, ..I*n*)、*using-column-list* が (U1, U2, ..U*n*) である場合、VALUES 句のない INSERT は INSERT (I1, I2, .. In) VALUES (U1, U2, .. Un) となります。

○ **RAISERROR**　RAISERROR は、マージ操作を終了し、変更をロールバックしてエラーを返すときに指定します。RAISERROR を指定すると、データベースサーバーはデフォルトで SQLSTATE 23510 および SQLCODE -1254 を返します。必要に応じて、RAISERROR キーワードの後に *error-number* パラメーターを指定し、返される SQLCODE をカスタマイズできます。カスタム SQLCODE には 17000 よりも大きい正の整数を指定してください。数または変数のいずれとしても指定できます。カスタム SQLCODE を指定すると、負の数が返されます。

たとえば、WHEN NOT MATCHED AND search-condition THEN RAISERROR 17001 を指定すると、WHEN 句の条件を満たすローが検出された場合、マージ操作は失敗し、変更がロールバックされ、SQLSTATE 23510 と SQLCODE -17001 を持つエラーが返されます。

○ **SKIP**　SKIP は、ローをスキップするときに指定します。何のアクションも行われません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

OPTION 句　この句は、文を実行するためのヒントを指定するときに使用します。次のヒントがサポートされます。

○ MATERIALIZED VIEW OPTIMIZATION *option-value*
○ FORCE OPTIMIZATION
○ *option-name* = *option-value* クエリテキスト内の OPTION(isolation_level = ...) の指定は、クエリの独立性レベルを指定する他のいずれの手段よりも優先されることに注意してください。

備考

source-object 内のローは *target-object* 内のローと比較され、ON 句の条件を満たすかどうかに応じて一致または不一致となります。*merge-search-condition* が true になるローが少なくとも 1 つ *target-table* に存在する場合、*source-object* のローは一致と見なされます。AND 句で指定された探索条件に従い、一致するローと一致しないローは WHEN MATCHED 句と WHEN NOT MATCHED 句で定義されたアクションによってグループ分けされます。一致するアクションと一致しないアクションごとにローをグループ化することを「分岐化」と呼び、各グループを「分岐」と呼びます。

分岐化が完了すると、データベースは分岐のローに定義されたアクションの実行を開始します。分岐は、出現した順序で処理されます。これは、文の中に WHEN 句が出現する順序と一致します。分岐化中、*source-object* の複数のローに、*target-object* の同じローに定義されたアクションがある場合、マージ操作は失敗してエラーが返されます。このことによって、*target-object* の指定されたローでマージ操作が複数のアクションを実行することがなくなります。

分岐が処理されると、挿入、更新、および削除の各アクションがそれぞれ INSERT 文、UPDATE 文、および DELETE 文としてトランザクションログに記録されます。

パーミッション

DBA 権限または次の権限が必要です。

●MERGE 文で INSERT、UPDATE、または DELETE の各アクションを指定する場合は、*target-object* の INSERT、UPDATE、および DELETE パーミッション。

●MERGE 文で参照されるすべてのオブジェクトに対する SELECT パーミッション。

●MERGE 文で参照されるすべてのプロシージャーに対する EXECUTE パーミッション。

関連する動作

target-object にトリガーが定義されていると、そのトリガーが起動されます。

参照

- 「MERGE 文を使用したデータのインポート」『SQL Anywhere サーバー SQL の使用法』
- 「UPDATE 文」976 ページ
- 「INSERT 文」802 ページ
- 「DELETE 文」697 ページ
- 「SELECT 文」895 ページ
- 「探索条件」34 ページ
- 「テーブル '%1' の MERGE 文は失敗しました。文で RAISERROR が指定されています」『エラーメッセージ』
- 「RAISERROR アクションの使用」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　MERGE 文は、SQL/2008 標準の機能 F312 および F313 を構成します。SQL Anywhere の MERGE 文は、SQL/2008 標準の MERGE 文仕様に準拠しており、追加の拡張が行われています。MERGE 文の SQL Anywhere 固有の拡張を次に示します。

  ○WHEN MATCHED 句の中の DELETE

  ○WHEN [NOT] MATCHED 句の中の RAISERROR

  ○WHEN [NOT] MATCHED 句の中の SKIP

  ○OPTION 句

  ○PRIMARY KEY 句

  ○DEFAULTS 句

  ○VALUES 句を指定しない INSERT 句

  ○WITH AUTO NAME 句

  ○SET 句を指定しない UPDATE 句

例

次の例は、派生テーブルのローを Products テーブルにマージします。この例は、既存の T シャツと同じ属性で、色、数量、および製品識別子の異なる新しい T シャツを追加します。ここでは、ID 番号 304 の製品が Products テーブルにすでに存在すると、ローは挿入されません。

```
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;
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

次の例は、前の例と同じですが、構文上の省略形を使用しません。

```
MERGE INTO Products ( ID, Name, Description, Size, Color, Quantity, UnitPrice, Photo )
  USING (
    SELECT 304 AS ID,
           'Purple' AS Color,
           100 AS Quantity,
           Name,
           Description,
           Size,
           UnitPrice,
           Photo
        FROM Products WHERE Products.ID = 300 )
        AS DT ( ID, Name, Description, Size, Color, Quantity, UnitPrice, Photo )
  ON ( Products.ID = DT.ID )
  WHEN NOT MATCHED
    THEN INSERT ( ID, Name, Description, Size, Color, Quantity, UnitPrice, Photo )
    VALUES ( DT.ID, DT.Name, DT.Description, DT.Size, DT.Color, DT.Quantity, DT.UnitPrice,
DT.Photo );
```

MESSAGE 文

メッセージを表示します。

構文

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*

パラメーター

TYPE 句　この句は、メッセージタイプを指定します。クライアントアプリケーションでは、メッセージの処理方法を指定します。たとえば、TO CLIENT を指定すると、Interactive SQL は、次のロケーションにメッセージを表示します。

○ **INFO**　**[メッセージ]** タブ。INFO がデフォルトです。

○ **ACTION**　**[OK]** ボタンのあるウィンドウ。

○ **WARNING**　**[OK]** ボタンのあるウィンドウ。

○ **STATUS**　**[メッセージ]** タブ。

TO 句　この句は、メッセージの送信先を指定します。

○ **CONSOLE**　データベースサーバーメッセージウィンドウにメッセージを送信します。データベースサーバーメッセージログファイルが指定されている場合は、このファイルにも送信します。CONSOLE がデフォルトです。

○ **CLIENT**　メッセージをクライアントアプリケーションに送信します。アプリケーションではメッセージの処理方法を決めます。また、この決定のベースとなる情報として TYPE を使用できます。

○ **LOG**　メッセージを -o オプションで指定されたデータベースサーバーメッセージログファイルに送信します。EVENT または SYSTEM を指定すると、メッセージはデータベースサーバーメッセージウィンドウ、Windows イベントログ (イベントソース SQLANY 12.0 Admin の下)、UNIX SysLog (SQLANY 12.0 Admin (servername) という名前の下) にも書き込まれます。データベースサーバーメッセージログのメッセージは以下のように指定します。

- **i**　タイプ INFO または STATUS のメッセージ。
- **w**　タイプ WARNING のメッセージ。
- **e**　タイプ ACTION のメッセージ。

FOR 句　TO CLIENT が指定されたメッセージの場合、この句はメッセージに関する通知を受信する接続を指定します。デフォルトでは、接続は次回 SQL 文または WAITFOR DELAY 文が実行されたときにメッセージを受信します。

○ **CONNECTION *conn-id-number***　受信者の接続 ID 番号を指定します。IMMEDIATE を指定すると、SQL 文がいつ実行されるかに関係なく、接続はメッセージを数秒以内に受信します。

○ **ALL**　開いているすべての接続がメッセージを受信することを指定します。

DEBUG ONLY　この句を使用すると、デバッグメッセージがストアドプロシージャーに追加されるかどうか、また debug_messages オプションの設定を変更することによってトリガーを有効にするか無効にするかを制御できます。DEBUG ONLY が指定されている場合、MESSAGE 文は、debug_messages オプションが On に設定されている場合にのみ実行されます。

> **注意**
> debug_messages オプションを Off に設定している場合、DEBUG ONLY メッセージはパフォーマンスへの影響が小さいため、通常は運用システム上のストアドプロシージャーに残すことができます。ただし、頻繁に実行する位置では控え目に使用してください。そうしないと、パフォーマンスがわずかに低下することがあります。

備考

MESSAGE 文は、メッセージを表示します。これには、任意の式を指定できます。句は、メッセージタイプおよびメッセージが表示される場所を指定します。

expression のサイズがデータベースページサイズを超えた場合、*expression* はデータベースページサイズに収まるようにトランケートされます。データベースに有効なページサイズをチェックするには、PageSize データベースプロパティに対するクエリを実行します (SELECT DB_PROPERTY('PageSize');)。

MESSAGE...TO CLIENT 文を実行するプロシージャーは、接続に関連している必要があります。

たとえば、次の例ではイベントが接続以外で発生しているため、ウィンドウは表示されません。

```
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
  MESSAGE 'Idle server' TYPE WARNING TO CLIENT;
END;
```

ただし、次の例では、データベースサーバーメッセージウィンドウにメッセージが書き込まれます。

```
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
  MESSAGE 'Idle server' TYPE WARNING TO CONSOLE;
END;
```

有効な式には、引用文字列または他の定数、変数、関数を含めることができます。

FOR 句を使用すると、アプリケーションでイベントを明示的にチェックしなくても、データベースサーバー上で検出されたイベントを別のアプリケーションに通知できます。FOR 句が使用されている場合、受信者は、SQL 文を次に実行するときにメッセージを受信します。受信者が現在 SQL 文を実行している場合は、文の完了時にメッセージが受信されます。実行されている文がストアドプロシージャー呼び出しである場合、メッセージは呼び出しが完了する前に受信されます。

アプリケーションがメッセージの送信直後に通知を必要とし、接続が SQL 文を実行していない場合は、複数の WAITFOR DELAY 文を同時に使用するのではなく、IMMEDIATE 句を使用してクライアント通知を実装します。

通常、宛先の接続がデータベースサーバー要求を実行していない場合でも、IMMEDIATE 句を使用して送信されたメッセージは 5 秒未満で配信されます。クライアント接続が 1 秒間に複数の要求を実行した場合、クライアント接続が非常に大きな BLOB データを受信した場合、またはクライアントのメッセージコールバックの実行に 1 秒以上かかった場合、メッセージの配信が遅れる可能性があります。また、複数の IMMEDIATE メッセージを 1 つの接続に 2 秒ごとに送信すると、メッセージの配信が遅れたり、エラーメッセージが生成されたりすることがあります。クライアント接続が切断されると、MESSAGE...IMMEDIATE 文が正常に配信されないことがあります。

IMMEDIATE 句のないメッセージが送信されるのは、クライアントが特定の要求または WAITFOR DELAY 文を実行した場合のみです。そのため、メッセージの配信時間は無制限になります。

IMMEDIATE 句には、SQL Anywhere 11 以降の DBLib、ODBC、または SQL Anywhere JDBC ドライバーが必要です。IMMEDIATE 句は、非スレッド型の UNIX クライアントライブラリではサポートされません。IMMEDIATE 句をサポートしない宛先接続にメッセージが送信されると、エラーメッセージが生成されます。MESSAGE 文の実行元と同じ接続に IMMEDIATE メッセージが送信されると、エラーメッセージが生成されます。

```
MESSAGE 'Please disconnect' TYPE WARNING TO CLIENT
  FOR CONNECTION 16 IMMEDIATE;
```

MESSAGE...TO CLIENT 式は、2048 バイトにトランケートされます。IMMEDIATE 句が指定されて送信されたメッセージの場合、メッセージ式は、接続のパケットサイズと 2048 バイトのうちの小さい方にトランケートされます。

Embedded SQL と ODBC クライアントは、メッセージコールバック関数を介してメッセージを受信します。それぞれの場合に、これらの関数を登録してください。Embedded SQL では、メッセージのコールバックが DB_CALLBACK_MESSAGE パラメーターを使用して db_register_a_callback に登録されます。ODBC では、メッセージのコールバックは、SA_REGISTER_MESSAGE_CALLBACK パラメーターを使用して SQLSetConnectAttr に登録されます。

パーミッション

FOR 句、TO EVENT LOG 句、または TO SYSTEM LOG 句を含む MESSAGE 文を実行するには、DBA 権限が必要です。

関連する動作

なし

参照

- 「sa_conn_info システムプロシージャー」1055 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「debug_messages オプション」『SQL Anywhere サーバー データベース管理』
- 「db_register_a_callback 関数」『SQL Anywhere サーバー プログラミング』
- 「WAITFOR 文」985 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次のプロシージャーは、データベースサーバーメッセージウィンドウにメッセージを表示します。

```
CREATE PROCEDURE message_text( )
BEGIN
MESSAGE 'The current date and time: ', Now( );
END;
```

次の文は、データベースサーバーメッセージウィンドウに文字列 The current date and time を表示し、続けて現在の日付と時刻を表示します。

```
CALL message_text( );
```

OPEN 文 [ESQL] [SP]

事前に宣言したカーソルを開き、データベースの情報にアクセスします。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 1 [ESQL]

OPEN *cursor-name*
[**USING** { **DESCRIPTOR** *sqlda-name* | *hostvar*, ... }]
[**WITH HOLD**]
[**ISOLATION LEVEL** *isolation-level*]
[**BLOCK** *n*]

構文 2 [SP]

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**

パラメーター

USING DESCRIPTOR 句　USING DESCRIPTOR 句を使用できるのは、Embedded SQL のみです。この句は、カーソルが宣言されている SELECT 文のプレースホルダーバインド変数にバインドされるホスト変数を指定します。

OPEN...USING はストアドプロシージャーでは使用できません。

WITH HOLD 句　デフォルトで、すべてのカーソルは現在のトランザクションの最後 (COMMIT または ROLLBACK) で自動的に閉じられます。オプションの WITH HOLD 句は、次に実行されるトランザクションのためにカーソルを開いたままにします。カーソルは現在の接続が終了するまで、または明示的な CLOSE 文が実行されるまで開いたままです。接続が終了すると、カーソルは自動的に閉じます。

COMMIT または ROLLBACK を実行すると、WITH HOLD カーソルの結果セットを構成するローを含め、接続で保持されている長期間のローロックがすべて解放されます。ただし、独立性レベル 1、2、3 で取得されたカーソル安定性ロックは、カーソルが存在するかぎり保持され、カーソルが閉じられるか、または接続が終了したときにのみ解放されます。

ROLLBACK 文が完了すると、WITH HOLD カーソルの内容とその範囲内での位置付けは予測できなくなり、保証されなくなります。ansi_close_cursors_on_rollback オプションを使用して、ROLLBACK 文によって WITH HOLD カーソルを自動的に閉じるかどうかを制御できます。

ISOLATION LEVEL 句　ISOLATION LEVEL 句を使用すると、isolation_level オプションの現在の設定とは異なる独立性レベルでカーソルを開くことができます。このカーソルのすべてのオペレーションを、オプション設定とは関係なく、指定した独立性レベルで実行できます。この句を指定しない場合、カーソルが開いている間のカーソルの独立性レベルは、カーソルを開いたときの isolation_level オプションの値です。

次の値がサポートされます。

○ 0
○ 1
○ 2
○ 3
○ SNAPSHOT
○ STATEMENT SNAPSHOT
○ READONLY STATEMENT SNAPSHOT

カーソルは、最初のローの前に置かれます。

BLOCK 句　この句は、Embedded SQL でのみ使用されます。クライアントアプリケーションは一度に複数のローをフェッチできます。これはブロックフェッチ、プリフェッチ、またはマルチローフェッチと呼ばれます。BLOCK 句を指定すると、プリフェッチされるローの数を減らすことができます。OPEN で BLOCK 句を指定することは、各 FETCH で BLOCK 句を指定することと同じです。

備考

OPEN 文は、指定したカーソルを開きます。カーソルを事前に宣言しておきます。

カーソルタイプが、カーソルの基本となる文の特性と一致していない場合、OPEN 文によって SQL 警告が返されることがあります。

カーソルが CALL 文にあるとき、OPEN は最初の結果セット (INTO 句がない SELECT 文) が見つかるまでプロシージャーを実行します。プロシージャーが完了しても結果セットが見つからない場合は、SQLSTATE_PROCEDURE_COMPLETE 警告が設定されます。

● **ESQL の使用方法**　OPEN 文が正常に実行された後で、SQLCA (SQLIOESTIMATE) の *sqlerrd*[*3*] フィールドに、クエリのすべてのローをフェッチするのに必要な入出力操作の数の推定値が格納されます。同様に、SQLCA (SQLCOUNT) の *sqlerrd*[*2*] フィールドに、カーソル内にある実際のローの数 (0 以上の値)、またはその推定値 (絶対値が推定値である負の数) が入ります。データベースサーバーによって計算されたローの数は、ローの実際の数です。ローを数える必要はありません。ローの実際の数を常に返すようにデータベースを設定することもできますが、これはおすすめしません。

  cursor-name を識別子または文字列によって指定する場合、C プログラムでは OPEN の前に対応する DECLARE CURSOR 文を置きます。*cursor-name* をホスト変数によって指定する場合、DECLARE CURSOR 文を OPEN 文の前で実行します。

パーミッション

SELECT 文のすべてのテーブルに SELECT パーミッション、または CALL 文のプロシージャーに EXECUTE パーミッションが必要です。

関連する動作

OPEN 文を実行すると、INSENSITIVE カーソルの結果セットが完全に実体化されます。

アクセスプランのキャッシュが有効になっている場合、OPEN 時にアプリケーションに返される SQL 警告が抑制されることがあります。抑制される警告には、カーソルタイプが変更された

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

こと、基本となるクエリが決定的でないこと、または文に埋め込まれている複数のリテラル定数で文字列のトランケーションが発生していることを示す警告があります。

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「RESUME 文」881 ページ
- 「PREPARE 文 [ESQL]」856 ページ
- 「FETCH 文 [ESQL] [SP]」749 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「RESUME 文」881 ページ
- 「CLOSE 文 [ESQL] [SP]」510 ページ
- 「FOR 文」753 ページ
- 「ansi_close_cursors_on_rollback オプション」『SQL Anywhere サーバー データベース管理』
- 「プランのキャッシュ」『SQL Anywhere サーバー SQL の使用法』
- 「insensitive カーソル」『SQL Anywhere サーバー プログラミング』
- 「row_counts オプション」『SQL Anywhere サーバー データベース管理』
- 「close_on_endtrans オプション」『SQL Anywhere サーバー データベース管理』
- 「Embedded SQL でのカーソル」『SQL Anywhere サーバー プログラミング』
- 「プロシージャー、トリガー、ユーザー定義関数、バッチのカーソル」『SQL Anywhere サーバー SQL の使用法』
- 「ロックの仕組み」『SQL Anywhere サーバー SQL の使用法』
- 「ロック期間」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　Embedded SQL 内での OPEN 文の使用は、オプションの SQL 言語機能 B031、"Basic dynamic SQL" の一部です。ストアドプロシージャー内での OPEN 文の使用は、SQL/2008 のコア機能です。ISOLATION LEVEL 句と BLOCK 句は、CALL 文でのカーソルの OPEN 機能と同様に、ベンダー拡張です。SQL/2008 標準では、WITH HOLD は、DECLARE CURSOR 文の一部として指定し、OPEN では指定しません。

  SQLCA の特定の値の設定はベンダー拡張です。

- **Transact-SQL**　OPEN 文は Adaptive Server Enterprise でサポートされています。Adaptive Server Enterprise では、ISOLATION LEVEL 句、BLOCK 句、WITH HOLD 句はサポートされていません。

例

次の例は、Embedded SQL での OPEN の使用を示します。

```
EXEC SQL OPEN employee_cursor;
```

と

```
EXEC SQL PREPARE emp_stat FROM
'SELECT empnum, empname FROM Employees WHERE name like ?';
EXEC SQL DECLARE employee_cursor CURSOR FOR emp_stat;
EXEC SQL OPEN employee_cursor USING :pattern;
```

次の例はプロシージャーまたはトリガーからのものです。

```
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
```

OUTPUT 文 [Interactive SQL]

現在のクエリ結果をファイルまたは ODBC データソースに出力します。

構文 1 - ファイルへの出力

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
| **FIXED**
| **HTML**
| **SQL**
| **XML**

encoding : *string* | *identifier*

構文 2 - ODBC データソースへの出力

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* [; ...]] }

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

APPEND 句　これはオプションのキーワードであり、既存の出力ファイルの前の内容を上書きしないで、末尾にクエリの結果を追加するために使用します。APPEND 句を使用しない場合、OUTPUT 文はデフォルトで出力ファイルの内容を上書きします。出力フォーマットが TEXT、FIXED、または SQL の場合に、APPEND キーワードが有効です。

BYTE ORDER MARK 句　この句は、Unicode ファイルの最初にバイトオーダーマーク (BOM) を追加するかどうかを指定するときに使用します。デフォルトでは、このオプションは ON です。この場合、ファイルの最初でバイトオーダーマーク (BOM) を書き込むように Interactive SQL に指示します。BYTE ORDER MARK が OFF の場合、DBISQL は BOM を書き込みません。

BYTE ORDER MARK 句は、TEXT フォーマットされたファイルに書き込む場合にのみ関係します。TEXT 以外の FORMAT 句とともに BYTE ORDER MARK 句を使用すると、エラーが返されます。

BYTE ORDER MARK 句は、UTF-8 か UTF-16 でエンコードされたファイルの読み込みまたは書き込みをするときにのみ使用されます。その他のエンコードで BYTE ORDER MARK 句を使用しようとすると、エラーが返されます。

COLUMN WIDTHS 句　COLUMN WIDTHS 句を使用して、FIXED フォーマット出力のカラム幅を指定します。

CREATE TABLE 句　CREATE TABLE 句を使用して、インポート先テーブルが存在しない場合に作成するかどうかを指定します。デフォルトは ON です。

DELIMITED BY 句　DELIMITED BY 句を使用できるのは、TEXT 出力フォーマットの場合のみです。カラムはデリミター文字列で区切られます。デフォルトはカンマです。

ENCODING 句　ENCODING 句では、ファイルの書き込みに使用されるエンコードを指定できます。ENCODING 句は、TEXT フォーマットでのみ使用できます。

ENCODING 句は、オペレーティングシステムの文字セットで表すことができないデータがある場合に役立ちます。この場合、ENCODING 句を使用しないと、デフォルトエンコードで表すことができない文字は出力から失われます (つまり、損失を伴う変換が発生します)。

OUTPUT 文を使用して入力ファイルを作成したときにエンコードを指定した場合は、INPUT 文でも同じ ENCODING 句を指定してください。

Interactive SQL を実行するとき、データのエクスポートに使用されるエンコードは次の順序で決定されます。

○ENCODING 句で指定されたエンコード (この句が指定されている場合)。

○default_isql_encoding オプションで指定されたエンコード (このオプションが設定されている場合)。

○実行しているプラットフォームのデフォルトエンコード。英語版 Windows コンピューターでは、デフォルトエンコードは 1252 です。

ESCAPE CHARACTER 句　16 進コードと記号として格納されている文字に使用するデフォルトのエスケープ文字は円記号 (¥) です。たとえば、¥x0A は改行文字です。

エスケープ文字は、ESCAPE CHARACTER 句を使って変更することができます。たとえば、感嘆符 (!) をエスケープ文字として使用するには、次のように指定します。

```
... ESCAPE CHARACTER '!'
```

改行文字は '¥n' と指定できます。他の文字は、タブ文字は ¥x09 などのように、16 進 ASCII コードを使用して指定できます。2 つの円記号 (¥) は 1 つの円記号として解釈されます。円記号 (¥) の後に n、x、X、¥ 以外の文字がある場合、それらは別々の文字と解釈されます。たとえば、¥q は円記号と q と解釈されます。

ESCAPES 句　ESCAPES を ON (デフォルト) にすると、データベースサーバーによって円記号に続く文字が認識され、特殊文字として解釈されます。ESCAPES を OFF にすると、ソースデータに記載されているとおりに文字が書き込まれます。

FORMAT 句　FORMAT 句によって、出力のファイルフォーマットを指定できます。FORMAT 句を指定していない場合、output_format オプションで指定したフォーマットが使用されます。FORMAT 句を指定した場合、output_format オプションの設定は無視されます。デフォルトの出力フォーマットは TEXT です。使用できる出力フォーマットは、次のとおりです。

- **TEXT**　出力は、ファイルの 1 行に 1 つのローが格納されたテキストフォーマットファイルです。すべての値をカンマで区切り、文字列をアポストロフィ (一重引用符) で囲みます。デリミターと引用符文字列は、DELIMITED BY と QUOTE 句を使って変更できます。ALL を QUOTE 句の中で指定する場合は、(文字列だけではなく) すべての値を引用符で囲みます。TEXT がデフォルトの出力タイプです。

  他の 3 つの特別なシーケンスも使用できます。2 つの文字 ¥n は改行文字を表し、¥ は ¥ を表し、シーケンス ¥xDD は 16 進コード DD の文字を表します。

  TEXT に出力し、出力内に引用符や改行を含めない場合は、QUOTE '' ESCAPES OFF のように、引用符やエスケープをオフにします。

- **FIXED**　この出力は、それぞれのカラムが固定幅を持つ固定フォーマットです。各カラムの幅は COLUMN WIDTHS 句を使用して指定できます。カラムの見出しはこのフォーマットでは出力されません。

  COLUMN WIDTHS 句を省略すると、各カラムの幅はデータ型から計算され、そのデータ型のどのような値でも十分に保持できる大きさになります。例外は、32 KB がデフォルトの LONG VARCHAR と LONG BINARY データです。

- **HTML**　この出力は HTML (Hyper Text Markup Language) フォーマットです。

- **SQL**　出力は、(テーブル内の情報を再作成するのに必要な) Interactive SQL の INPUT 文で、*.sql* ファイルに格納されます。

- **XML**　この出力は、UTF-8 でエンコードされ、DTD が埋め込まれた XML ファイルです。バイナリ値は、2 桁の 16 進数文字列として表されるバイナリデータとして CDATA ブロック内にエンコードされます。

HEXADECIMAL 句　HEXADECIMAL 句は、バイナリ値を TEXT フォーマットに出力する方法を指定します。使用できる値は、次のとおりです。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **ON**　ON に設定すると、0x プレフィクスとその後に続く一連の 16 進ペアでバイナリ値が書き込まれます。たとえば、0xabcd のようになります。

○ **OFF**　OFF に設定すると、印刷不能な文字値の先頭にエスケープ文字 (円記号など) が追加され、その後に x、バイトの 16 進ペアと続きます。印刷可能な文字はそのまま出力されます。

たとえば、次のコマンドは、'one¥x0Atwo¥x0Athree' を含むファイルを出力します。

```
SELECT 'one¥ntwo¥nthree'
OUTPUT TO 'test.txt' HEXADECIMAL OFF;
```

○ **ASIS**　ASIS に設定すると、値はそのまま書き込まれます。値が制御文字を含む場合も、エスケープはされません。ASIS は、タブや改行などのフォーマット記号を含むテキストに適しています。

QUOTE 句　QUOTE 句を使用できるのは、TEXT 出力フォーマットの場合のみです。文字列値は引用符で囲みます。デフォルトは一重引用符 (') です。ALL を QUOTE 句の中に指定する場合、引用符文字列を文字列だけではなくすべての値の周囲に配置します。引用符で囲まないようにするには、空の一重引用符を指定します。たとえば、QUOTE '' のように指定します。

USING 句　USING 句は ODBC データソースにデータをエクスポートします。DSN オプションを使用して ODBC データソース名を指定するか、または DRIVER オプションを使用して ODBC ドライバー名と接続パラメーターを指定できます。*Connection-parameter* は、データベース特定の接続パラメーターのオプションのリストです。

Odbc-data-source は、ユーザー名または ODBC データソース名です。たとえば、SQL Anywhere サンプルデータベースの *odbc-data-source* は SQL Anywhere 12 Demo です。

Odbc-driver-name は ODBC ドライバー名です。SQL Anywhere データベースの場合、*odbc-driver-name* は SQL Anywhere です。Ultra Light データベースの場合、*odbc-driver-name* は Ultra Light 12 です。

VERBOSE 句　オプションの VERBOSE キーワードを指定すると、クエリに関するエラーメッセージ、データの選択に使用された SQL 文、データ自体が出力ファイルに書き込まれます。データを含まない行には 2 つのハイフン (--) のプレフィクスが付きます。VERBOSE を省略すると (デフォルト)、ファイルにはデータのみが書き込まれます。出力フォーマットが TEXT、FIXED、または SQL の場合に、VERBOSE キーワードが有効です。

WITH COLUMN NAMES 句　WITH COLUMN NAMES 句は、テキストファイルの先頭行にカラム名を挿入します。WITH COLUMN NAMES 句を使用できるのは、TEXT フォーマットの場合のみです。

備考

OUTPUT 文は、ファイルまたはデータベースにデータを出力します。出力するデータを取り出す文の直後に OUTPUT 文を使用します。

複数の結果セットをエクスポートするには、構文 1 を使用し、isql_show_multiple_result_sets オプションを On に設定します。Interactive SQL では、結果セットごとにファイルが作成されます。ファイルの名前は *filename-x* のようになります。ここで、x は 1 から始まるカウンターです。たとえば、OUTPUT TO で *data.txt* という名前のファイルを指定すると、*data-1.txt*、*data-2.txt* のようなファイル名が使用されます。

構文 2 を使用して複数の結果セットをエクスポートすることはできません。

出力フォーマットは、オプションの FORMAT 句を使って指定できます。デフォルトのフォーマットは TEXT です。FORMAT 句を指定しない場合、Interactive SQL output_format オプションの設定が使用されます。

OUTPUT 文は Interactive SQL コマンドであるため、複合文 (IF など) やストアドプロシージャーでは使用できません。

BINARY データまたは LONG BINARY データを含むカラムを Microsoft Excel ブックにエクスポートする場合は、データを文字列または数値に変換してください。また、データを Microsoft Excel ブックにエクスポートする場合、DSN オプションの選択時に ReadOnly パラメーターを 0 に設定するか、オフにしないと、データは読み込み専用になります。

パーミッション

なし

関連する動作

Interactive SQL の場合、**[結果]** タブには現在のクエリの結果が表示されます。

参照

- 「SELECT 文」895 ページ
- 「INPUT 文 [Interactive SQL]」794 ページ
- 「UNLOAD 文」966 ページ
- 「output_format オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「データのインポートとエクスポート」『SQL Anywhere サーバー SQL の使用法』
- 「default_isql_encoding オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「isql_show_multiple_result_sets オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』
- 「OUTPUT 文を使用したデータのエクスポート」『SQL Anywhere サーバー SQL の使用法』
- 「プロシージャー、トリガー、イベント、バッチで使用できる文」『SQL Anywhere サーバー SQL の使用法』
- 「プロシージャーから複数の結果セットを返す」『SQL Anywhere サーバー SQL の使用法』
- 「output_format オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「サポートされている文字セット」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

Employees テーブルの内容をテキストファイルに出力します。

```
SELECT *
FROM Employees;
OUTPUT TO 'Employees.txt'
FORMAT TEXT;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Employees テーブルの内容を既存のテキストファイルの最後に追加し、クエリに関するメッセージも同じファイルに追加します。

```
SELECT *
FROM Employees;
OUTPUT TO 'Employees.txt'
APPEND VERBOSE;
```

改行文字が埋め込まれた値をエクスポートする必要があるとします。改行文字の数値は 10 です。SQL 文ではこれを文字列 '¥x0a' として表すことができます。たとえば、HEXADECIMAL を ON に設定して次の文を実行します。

```
SELECT CAST ('line1¥x0aline2' AS VARBINARY);
OUTPUT TO 'file.txt' HEXADECIMAL ON;
```

次のテキストを含む 1 行のファイルができます。

```
0x6c696e65310a6c696e6532
```

同じ文を、HEXADECIMAL を OFF にして実行すると、次のようになります。

```
'line1¥x0Aline2'
```

最後に、HEXADECIMAL を ASIS に設定すると、2 行のファイルができます。

```
'line1
line2'
```

ASIS を使用した場合に 2 行になるのは、埋め込まれた改行文字が 2 桁の 16 進表現に変換されず、プレフィクスも付かないままエクスポートされたためです。

次の例は、Customers テーブルから新しいテーブル Customers2 にデータを出力します。

```
SELECT * FROM Customers;
OUTPUT USING 'dsn=SQL Anywhere 12 Demo'
INTO "Customers2";
```

次の例は、DRIVER オプションを使用して、サンプルデータベースからデータベース *mydatabase.db* に Customers テーブルをコピーします。

```
SELECT * FROM Customers;
OUTPUT USING 'DRIVER=SQL Anywhere 12;uid=dba;pwd=sql;dbf=c:¥test¥mydatabase.db'
INTO "Customers";
```

次の例は、DRIVER オプションを使用して、SQL Anywhere サンプルデータベースから架空の Ultra Light データベース *myULDatabase.db* の Customers テーブルに Customers テーブルをコピーします。

```
SELECT * FROM Customers;
OUTPUT USING 'DRIVER=UltraLite 12;dbf=c:¥test¥myULDatabase.udb'
INTO "Customers";
```

次の例は、DRIVER オプションを使用して、架空の MySQL データベース *mydatabase* に Customers テーブルをコピーします。

```
SELECT * FROM Customers;
OUTPUT USING 'DRIVER=MySQL ODBC 5.1
```

```
Driver;DATABASE=mydatabase;SERVER=mySQLHost;UID=me;PWD=secret'
INTO "Customers";
```

PARAMETERS 文 [Interactive SQL]

Interactive SQL スクリプトファイルにパラメーターを指定します。

構文

PARAMETERS *parameter1*, *parameter2*, ...

備考

PARAMETERS 文はスクリプトファイルのパラメーターに名前を付けて、スクリプトファイルの中で後から参照できるようにします。

パラメーターを参照するには、指定したパラメーターを置き換えるファイルの中に {parameter1} を配置します。大カッコとパラメーター名の間には、スペースを入れないでください。

スクリプトファイルを呼び出すときに、すべての必要なパラメーターを指定しないと、Interactive SQL は不足しているパラメーターの値を要求するメッセージを表示します。

パーミッション

なし

関連する動作

なし

参照

- 「READ 文 [Interactive SQL]」 863 ページ
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の Interactive SQL スクリプトファイルは、2 つのパラメーターを取ります。

```
PARAMETERS department_id, file;
SELECT Surname
FROM Employees
WHERE DepartmentID = {department_id}
>#{file}.dat;
```

このスクリプトをファイル *test.sql* に保存すると、次のコマンドを使用して Interactive SQL から実行できます。

```
READ test.sql [100] [data]
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

PASSTHROUGH 文 [SQL Remote]

SQL Remote 管理のパススルーモードを起動または停止します。構文 1 と 2 はパススルーモードを起動し、構文 3 はパススルーモードを停止します。

構文 1

PASSTHROUGH [**ONLY**] **FOR** *userid*, ...

構文 2

PASSTHROUGH [**ONLY**] **FOR SUBSCRIPTION**
TO [*owner*.]*publication-name* [**(** *constant* **)**]

構文 3

PASSTHROUGH STOP

備考

パススルーモードでは、SQL 文はすべてデータベースサーバーで実行されます。また、トランザクションログに保存され、メッセージに含めてサブスクライバーに送信されます。パススルーモードの起動に ONLY キーワードを使用した場合、この文はサブスクライバーに送られるだけで、サーバーでは実行されません。パススルーセクションにストアドプロシージャーへの呼び出しが含まれている場合、そのストアドプロシージャーは、パススルーコマンドを発行するサーバー上に存在していなければなりません (これらのストアドプロシージャーの中には、サーバー上では実行されないものもあります)。パススルー SQL 文の受信者は、ユーザー ID リスト (構文 1)、または指定されたパブリケーションのすべてのサブスクライバーです。パススルーモードは、統合データベースからリモートデータベースへの変更を適用したり、リモートデータベースから統合データベースへ文を送信するのにも使用できます。

構文 2 は、サブスクリプションを起動したリモートデータベースに文を送信します。サブスクリプションを作成しても、起動していないリモートデータベースには送信しません。

構文 3 は、現在の接続のパススルーモードを停止します。PASSTHROUGH STOP 文は、パススルーモードを開始した接続で実行してください。構文 1 または 2 を使用して接続のパススルーモードを起動し、その接続が PASSTHROUGH STOP 文の実行前に切断されると、切断によって PASSTHROUGH STOP 文が暗黙的に実行されます。

パーミッション

DBA 権限。

関連する動作

なし

標準と互換性

- **SQL/2008** ベンダー拡張。

参照

- 「FORWARD TO 文」757 ページ
- 「UNLOAD 文」966 ページ
- 「接続のテスト」『SQL Anywhere サーバー SQL の使用法』
- 「SQL Remote のパススルーモード」『SQL Remote』
- 「パススルーモードの開始と停止」『SQL Remote』
- 「パススルーモードの制限事項」『SQL Remote』
- 「実行中のシステムに行ってはならない変更」『SQL Remote』
- 「アップグレードと再同期」『SQL Remote』
- 「サブスクリプションの再同期」『SQL Remote』

例

```
PASSTHROUGH FOR rem_db ;
...
( SQL statements to be executed at the remote database )
...
PASSTHROUGH STOP ;
```

PREPARE 文 [ESQL]

後で実行する文を準備するか、カーソルを定義します。

構文

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** }

パラメーター

statement-name 文の名前は、識別子またはホスト変数とすることができます。ただし、複数の SQLCA を使用する場合には識別子を使用しないでください。そのような場合に識別子を使用すると、2 つの準備文の文番号が同じになり、誤った文が実行されたり開かれたりすることがあります。また、マルチスレッドのアプリケーションの場合、文の名前に識別子を使用することはおすすめしません。文の名前は複数のスレッドから同時に参照される場合があるためです。

DESCRIBE 句 DESCRIBE INTO DESCRIPTOR を使用すると、準備文は指定した記述子に記述されます。記述タイプとして、DESCRIBE 文で許容されるいずれかのものを使用できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

FOR UPDATE | FOR READ ONLY　カーソルで文が使用される場合のカーソル更新可能性を定義します。FOR READ ONLY カーソルは UPDATE (位置付け) 操作や DELETE (位置付け) 操作には使用できません。FOR READ ONLY がデフォルトです。

FOR UPDATE を指定したカーソル要求への応答では、SQL Anywhere は value-sensitive カーソルまたは sensitive カーソルを提供します。insensitive カーソルと asensitive カーソルは更新できません。

BY VALUES | BY TIMESTAMP　データベースサーバーは、キーセット駆動型カーソルを利用して、結果セットをスクロールするときにローが変更または削除された場合にアプリケーションへ通知できるようにします。

BY LOCK 句　データベースサーバーは、結果セットのフェッチされたローに対する意図的ローロックを取得します。このロックは長期間のロックであり、トランザクションがコミットまたはロールバックされるまで保持されます。

- **WITH EXECUTE 句**　WITH EXECUTE 句を使用すると、CALL または SELECT 文ではなく、ホスト変数が含まれていない場合にのみ、文が実行されます。正常に実行された後、文はその場で削除されます。文の準備と記述が正常に終了し、文が実行できない場合、警告 SQLCODE 111、SQLSTATE 01W08 が設定され、文は削除されます。

  DESCRIBE INTO DESCRIPTOR 句と WITH EXECUTE 句を使用すると、必要なクライアント／サーバー通信を最小限に抑えて、パフォーマンスを改善できる場合があります。

- **WITH VARIABLE RESULT 句**　WITH VARIABLE RESULT 句は、複数の結果セットと異なる数または種類のカラムを持つプロシージャーの記述に使用します。

  WITH VARIABLE RESULT を使用する場合、記述後にデータベースサーバーによって SQLCOUNT 値に次のいずれかの値が設定されます。

  - **0**　結果セットは変更される場合があります。各 OPEN 文の後でプロシージャー呼び出しを再度記述してください。
  - **1**　結果セットは固定です。再度記述する必要はありません。

> **注意**
> 互換性を保つために、COMMIT、PREPARE TO COMMIT、ROLLBACK 文の準備は引き続きサポートされています。ただし、静的 Embedded SQL を使って、すべてのトランザクション管理操作を行うことをおすすめします。特定のアプリケーション環境では、これが必要とされるからです。また、他の Embedded SQL システムは、動的トランザクション管理操作をサポートしません。

備考

PREPARE 文は、*statement* から SQL 文を準備し、準備した文を *statement-name* と関連付けます。この文名を参照し、文を実行します。または、文が SELECT 文または CALL 文の場合、カーソルを開きます。*statement-name* は、*sqlca.h* ヘッダーファイルの中で定義される a_sql_statement_number 型のホスト変数であり、自動的に含まれます。識別子が *statement-name* に使用される場合、モジュールごとに 1 つの文だけが、*statement-name* と一緒に準備されます。

statement-name にホスト変数を使用する場合は、short int 型にします。*sqlca.h* 内には、この型の型定義 a_sql_statement_number があります。この型を SQL プリプロセッサーが認識し、DECLARE セクションの中で使用できます。ホスト変数は、PREPARE 文の実行中にデータベースによって定義されるため、初期化する必要はありません。

パーミッション

なし

関連する動作

以前に同じ名前で準備した文が失われます。

WITH EXECUTE を指定した文は、実行が成功した場合のみ削除されます。それ以外の場合は、DROP を使って文を使用後に削除してください。DROP しないと、文が使ったメモリを再使用できません。

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「DESCRIBE 文 [ESQL]」700 ページ
- 「OPEN 文 [ESQL] [SP]」844 ページ
- 「EXECUTE 文 [ESQL]」742 ページ
- 「DROP STATEMENT 文 [ESQL]」726 ページ
- 「文を実行することができませんでした。」『エラーメッセージ』
- 「動的 SQL 文」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　PREPARE は、オプションの SQL/2008 言語機能 B031、"Basic dynamic SQL" の一部です。オプションの FOR UPDATE、FOR READ ONLY、DESCRIBE、WITH EXECUTE 句はベンダー拡張です。

例

次の文は、単純なクエリを準備します。

```
EXEC SQL PREPARE employee_statement FROM
'SELECT Surname FROM Employees';
```

PREPARE TO COMMIT 文

COMMIT を正常に実行できるかどうかをチェックします。

構文

PREPARE TO COMMIT

備考

PREPARE TO COMMIT 文は、COMMIT がうまく実行できるかどうかをテストします。COMMIT を実行するときデータベースの整合性違反があると、文はエラーを発生します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

PREPARE TO COMMIT 文は、ストアドプロシージャー、トリガー、イベント、またはバッチに使用できません。

パーミッション

なし

関連する動作

なし

参照

- 「COMMIT 文」514 ページ
- 「ROLLBACK 文」890 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文のシーケンスは、外部キーが Employees テーブルをチェックするため、エラーになります。

```
EXECUTE IMMEDIATE
  "SET OPTION wait_for_commit = 'On'";
EXECUTE IMMEDIATE "DELETE FROM Employees
  WHERE EmployeeID = 160";
EXECUTE IMMEDIATE "PREPARE TO COMMIT";
```

次の一連の文では、削除文の実行時に整合性違反があっても、エラーは発生しません。PREPARE TO COMMIT 文はエラーを返します。

```
SET OPTION wait_for_commit= 'On';
DELETE
FROM Departments
WHERE DepartmentID = 100;
PREPARE TO COMMIT;
```

PRINT 文 [T-SQL]

クライアントにメッセージを返すか、データベースサーバーメッセージウィンドウにメッセージを表示します。

構文

PRINT *format-string* [, *arg-list*]

備考

PRINT 文は、Open Client アプリケーションまたは jConnect アプリケーションから接続している場合、クライアントのウィンドウにメッセージを返します。Embedded SQL または ODBC アプリケーションから接続している場合、このメッセージはデータベースサーバーメッセージウィンドウに表示されます。

フォーマット文字列は、オプション引数リスト (arg-list) にある引数のプレースホルダーを含むことができます。プレースホルダーの形式は *%nn!* で、*nn* は 1 ～ 20 の間の整数です。

パーミッション

なし

関連する動作

なし

参照

- 「MESSAGE 文」841 ページ

標準と互換性

- **SQL/2008** Transact-SQL 拡張。

例

次の文はメッセージを表示します。

```
PRINT 'Display this message';
```

次の文は、PRINT 文内でのプレースホルダーの使い方を示します。

```
DECLARE @var1 INT, @var2 INT
SELECT @var1 = 3, @var2 = 5
PRINT 'Variable 1 = %1!, Variable 2 = %2!', @var1, @var2
```

PUT 文 [ESQL]

指定したカーソルにローを挿入します。

構文

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 :インジケーター変数を含めることができる

row-count : *integer* | *hostvar*

備考

指定したカーソルにローを挿入します。カラムの値は、最初の SQLDA から取得されます。または、INSERT 文に指定したカラム (INSERT カーソル用) または SELECT リストのカラム (SELECT カーソル用) と 1 対 1 で対応するホスト変数リストから取得されます。

PUT 文は、INSERT 文または SELECT 文のカーソルでのみ使用でき、これらの文では FROM 句に指定した 1 つのテーブルを参照するか、1 つのベーステーブルで構成される更新可能ビューを参照します。

SQLDA 内の sqldata ポインターが NULL ポインターである場合、そのカラムには値を指定しません。sqldata ポインターが DEFAULT VALUE を持つ場合、カラムはこの値を使います。そうでない場合は、NULL 値を使います。

2 番目の SQLDA またはホスト変数のリストには、PUT 文の結果が格納されています。

オプションの ARRAY 句を使って、ワイドプットを実行します。ワイドプットでは一度に複数のローが挿入され、パフォーマンスが改善されます。整数値は、挿入するローの数です。SQLDA には各エントリの変数 (ロー数 * カラム数) を入れます。最初のローは SQLDA の変数 0 から (ロー当たりのカラム数) -1 に入り、以後のローも同様です。

注意
スクロール (value-sensitive) カーソルの場合は、新しいローが WHERE 句と一致し、キーセットカーソルが移植を完了していない場合に、挿入されたローが表示されます。動的カーソルの場合は、挿入されたローが WHERE 句と一致すると、そのローが表示される場合があります。Insensitive カーソルは更新できません。

パーミッション

INSERT パーミッションが必要です。

関連する動作

ローを value-sensitive (キーセット駆動型) カーソルに挿入するとき、挿入されたローは結果セットの最後に表示されます。ローがクエリの WHERE 句に一致しない場合も、ORDER BY 句が正常にローを結果セットの別の場所に置く場合も同様です。

参照

- 「UPDATE 文」976 ページ
- 「UPDATE (位置付け) 文 [ESQL] [SP]」971 ページ
- 「DELETE 文」697 ページ
- 「DELETE 文 (位置付け) [ESQL] [SP]」695 ページ
- 「INSERT 文」802 ページ
- 「SET 文」927 ページ
- 「ローの修正に使用するカーソル」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、Embedded SQL での PUT の使い方を示します。

```
EXEC SQL PUT cur_employee FROM :employeeID, :surname;
```

RAISERROR 文

エラー信号を送り、クライアントにメッセージを送信します。

構文

RAISERROR *error-number* [*format-string*] [, *arg-list*]

パラメーター

error-number *error-number* は、17000 を超える 5 桁の整数です。このエラー番号はグローバル変数 @@error に格納されます。

format-string *format-string* を指定しないか、空にすると、エラー番号を使用してシステムテーブルのエラーメッセージが検索されます。Adaptive Server Enterprise は、17000 から 19999 までのメッセージを SYSMESSAGES テーブルから取得します。SQL Anywhere では、このテーブルは空のビューなので、この範囲のエラーはフォーマット文字列を指定するようにしてください。エラー番号が 20000 以上のメッセージは、ISYSUSERMESSAGE システムテーブルから取得します。

新しいエラーメッセージを作成するには、CREATE ERROR 文を使用します。ISYSUSERMESSAGE システムテーブルのメッセージを表示するには、SYSUSERMESSAGE システムビューに問い合わせます。

SQL Anywhere では、*format-string* の長さは 255 バイトまで可能です。

Adaptive Server Enterprise RAISERROR 文でサポートされる拡張値は、SQL Anywhere ではサポートされていません。

フォーマット文字列は、オプション引数リスト (arg-list) にある引数のプレースホルダーを含むことができます。プレースホルダーの形式は %*nn*! で、*nn* は 1 ～ 20 の間の整数です。

中間レベルの RAISERROR のステータスとコード情報は、プロシージャーが終了すると失われます。結果が返されるときに RAISERROR と一緒にエラーが発生した場合は、エラー情報が返され、RAISERROR 情報は失われます。アプリケーションでは、別の実行ポイントでグローバル変数 @@error を検査して、中間の RAISERROR ステータスを問い合わせることができます。

備考

RAISERROR 文で、ユーザー定義エラーの信号を送り、クライアント上にメッセージを送信できます。

パーミッション

なし

関連する動作

なし

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「BEGIN 文」497 ページ
- 「CREATE MESSAGE 文 [T-SQL]」578 ページ
- 「SYSUSERMESSAGE システムビュー」1329 ページ
- 「CREATE TRIGGER 文 [T-SQL]」676 ページ
- 「CREATE TRIGGER 文」670 ページ
- 「on_tsql_error オプション」『SQL Anywhere サーバー データベース管理』
- 「login_procedure オプション」『SQL Anywhere サーバー データベース管理』
- 「continue_after_raiserror オプション」『SQL Anywhere サーバー データベース管理』
- 「SIGNAL 文 [SP]」934 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

- **Transact-SQL**　RAISERROR 文は Transact-SQL 複合文およびプロシージャーでは使用できません。

例

次の文はエラー 23000 を発します。これはユーザー定義エラーの範囲で、クライアントにメッセージを送信します。パラメーター *error-number* と *format-string* の間にカンマがないことに注意してください。カンマの後にある最初の項目は、引数リストの最初の項目として解釈されます。

```
RAISERROR 23000 'Invalid entry for this column: %1!', @val
```

次の例では、RAISERROR を使用して接続を禁止します。

```
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
```

READ 文 [Interactive SQL]

ファイルから Interactive SQL 文を読み込みます。

構文

READ [**ENCODING** *encoding*] *filename* **[** *parameter* **]** ...

encoding : *identifier* | *string*

パラメーター

- **ENCODING**　ENCODING 句では、ファイルの読み込みに使用されるエンコードを指定できます。READ 文は、ファイルを読み込むときにエスケープ文字を処理しません。ファイル全体が指定されたコードであると想定します。

  Interactive SQL を実行するとき、データの読み込みに使用されるエンコードは次の順序で決定されます。

  1. ENCODING 句で指定されたエンコード (この句が指定されている場合)。
  2. ファイル内でバイトオーダーマーク (BOM) で指定されたエンコード (BOM が指定されている場合)。
  3. default_isql_encoding オプションで指定されたエンコード (このオプションが設定されている場合)。
  4. 実行しているプラットフォームのデフォルトエンコード。英語版 Windows コンピューターでは、デフォルトエンコードは 1252 です。

備考

READ 文は、指定したファイルから Interactive SQL 文のシーケンスを読み込みます。このファイルには、別の READ 文を含む有効な Interactive SQL 文が含まれています。READ 文はどのような深さにまでもネストできます。

filename にファイル拡張子がない場合、Interactive SQL は拡張子 *.sql* を持つ同じファイル名を検索します。

filename に絶対パスが含まれていない場合は、Interactive SQL がファイルを検索します。*filename* のロケーションは、次のように READ 文のロケーションに基づいて決定されます。

- Interactive SQL で READ 文を直接実行した場合、Interactive SQL は、最初に *filename* へのパスを Interactive SQL が動作しているディレクトリとの相対パスとして解決しようとします。このことが失敗した場合、Interactive SQL は、環境変数 SQLPATH の中で指定されたディレクトリ内で *filename* を検索し、さらに環境変数 PATH の中で指定されたディレクトリ内で検索します。

- Interactive SQL では、READ 文が外部ファイル (*.sql* ファイルなど) に存在する場合、最初に *filename* へのパスを外部ファイルのロケーションとの相対パスとして解決しようとします。これが失敗した場合は、Interactive SQL が動作しているディレクトリとの相対パスで *filename* を検索します。引き続き失敗した場合、Interactive SQL は、環境変数 SQLPATH の中で指定されたディレクトリ内で検索し、さらに環境変数 PATH の中で指定されたディレクトリ内で検索します。

パラメーターは、SQL スクリプトファイル名の後にリストできます。これらのパラメーターは、文ファイルの先頭の PARAMETERS 文で指定したパラメーターに対応します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター名は角カッコで囲む必要があります。Interactive SQL は、ソースファイルに { parameter-name } がある場合、対応するパラメーターを置き換えます。この *parameter-name* は適切なパラメーター名です。

スクリプトファイルに渡すパラメーターには、識別子、数値、引用符付きの識別子、または文字列を指定できます。パラメーターの周囲を引用符で囲むときは、入れ替えるテキストの中に引用符を入れてください。識別子、数、または文字列 (スペースまたはタブを含む) ではないパラメーターは、角カッコ ([ ]) で囲みます。こうすると、スクリプトファイルの中で任意のテキストを置き換えることができます。

十分なパラメーターがスクリプトファイルに渡されない場合、Interactive SQL は足りないパラメーターの値を要求するメッセージを表示します。

Interactive SQL で *reload.sql* ファイルを実行する場合は、パラメーターとして暗号化キーを指定する必要があります。READ 文にキーを指定していない場合、Interactive SQL でキーの入力を要求するプロンプトが表示されます。

パーミッション

なし

関連する動作

なし

参照

- 「PARAMETERS 文 [Interactive SQL]」854 ページ
- 「default_isql_encoding オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL ユーティリティ (dbisql)」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次は、READ 文の例です。

```
READ status.rpt '160';
READ birthday.sql [>= '1988-1-1'] [<= '1988-1-30'];
```

特定の OEM コードページを使用するファイルを処理するには、そのコードページを指定する必要があります。次に例を示します。

```
dbisql READ ENCODING 'cp437' myfile.sql
```

READTEXT 文 [T-SQL]

指定したオフセットから開始して、指定したバイト数のテキスト値とイメージ値をデータベースから読み込みます。この機能は Transact-SQL との互換性のためにのみ提供されているものであり、使用しないことをおすすめします。

構文

READTEXT *table-name.column-name*
text-pointer-offset text-size
[**HOLDLOCK**]

備考

READTEXT は、データベースから CHAR、NCHAR、BINARY カラムを読み込むのに使用されます。ビューに対して READTEXT 操作を実行することはできません。

パーミッション

テーブルに対する SELECT パーミッションが必要です。

関連する動作

なし

参照

- 「WRITETEXT 文 [T-SQL]」991 ページ
- 「GET DATA 文 [ESQL]」771 ページ
- 「TEXTPTR 関数 [テキストとイメージ]」368 ページ

標準と互換性

- **SQL/2008**　Transact-SQL 拡張。

例

次の Embedded SQL の例では、TEXTPTR を使用して、MarketingInformation テーブルの ProductID 500 に関連付けられている Description カラムを見つけます。

テキストポインターは、変数 **txtptr** に格納され、READTEXT 文のパラメーターとして指定されます。READTEXT 文は、カラムオフセット 181 から開始して 55 バイトを返します。

```
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 );
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

READTEXT は次の文字列を返します。

Lightweight 100% organically grown cotton construction.

REFRESH MATERIALIZED VIEW 文

クエリ定義を実行することで、マテリアライズドビューのデータを初期化または更新します。

構文

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**

パラメーター

WITH 句　WITH 句は、基本となるベーステーブルでリフレッシュ中に使用するロックの種類を指定する場合に使用します。ロックの種類によって、マテリアライズドビューの移植方法とトランザクションの同時実行性への影響が決まります。WITH 句の設定は、マテリアライズドビュー自体にかけられるロックの種類には影響しません。このロックは常に排他ロックです。ロックに指定できる句は、次のいずれかです。

○ **ISOLATION LEVEL *isolation-level***　WITH ISOLATION LEVEL は、リフレッシュ操作を実行する場合の独立性レベルを変更する場合に使用します。元の独立性レベルは、文の完了時に接続に対してリストアされます。

即時ビューの場合、*isolation-level* に指定できるのは SERIALIZABLE のみです。

スナップショットアイソレーションの場合、トランザクションスナップショットレベルのみが REFRESH MATERIALIZED VIEW 文でサポートされています。独立性レベルとして SNAPSHOT を指定してください。statement-snapshot と readonly-statement-snapshot レベルはサポートされていません。

○ **EXCLUSIVE MODE**　WITH EXCLUSIVE MODE は、独立性レベルを変更しないが、基本となるテーブルにコミットされたデータと矛盾しないようにデータを確実に更新する場合に使用します。WITH EXCLUSIVE MODE を使用すると、基本となるすべてのベーステーブルに排他テーブルロックがかけられます。リフレッシュ操作が完了するまで、他のトランザクションによって、基本となるテーブルに対する問い合わせ、更新、その他のアクションは実行できなくなります。排他テーブルロックを取得できない場合、リフレッシュ操作は失敗し、エラーが返されます。

○ **SHARE MODE**　WITH SHARE MODE は、リフレッシュ操作の実行中に、基本となるテーブルを他のトランザクションで読み込めるようにするために使用します。この句を指定すると、リフレッシュ操作が実行される前から、リフレッシュ操作が完了するまで、基本となるすべてのベーステーブルの共有テーブルロックが取得されます。

FORCE BUILD 句　デフォルトでは、REFRESH MATERIALIZED VIEW 文を実行すると、データベースサーバーはマテリアライズドビューが古い (つまり、マテリアライズドビューが最後にリフレッシュされた後、基本となるテーブルが変更されている) かどうかをチェックします。マテリアライズドビューが古くない場合、リフレッシュは行われません。FORCE BUILD 句は、マテリアライズドビューが古いかどうかに関係なく、強制的にマテリアライズドビューをリフレッシュするときに指定します。

備考

この文は、*view-list* にリストされているマテリアライズドビューを初期化またはリフレッシュする場合に使用します。

古くなっていないマテリアライズドビューに対して REFRESH MATERIALIZED VIEW 文を実行した場合、FORCE BUILD 句が指定されていないかぎり、リフレッシュは実行されません。

ロックとデータ同時実行性の、デフォルトのリフレッシュ動作は次のとおりです。

- ビューが即時ビューの場合、スナップショットアイソレーションが有効かどうかに関係なく、デフォルトのリフレッシュ動作は WITH SHARE MODE です。
- ビューが手動ビューであり、スナップショットアイソレーションを**使用している**場合、デフォルトは WITH ISOLATION LEVEL SNAPSHOT です。
- ビューが手動ビューであり、スナップショットアイソレーションを**使用していない**場合、デフォルトは WITH SHARE MODE です。

REFRESH MATERIALIZED VIEW が正常に動作する場合、および、最適化にビューを使用する場合は、いくつかのオプションに特定の値が設定されている必要があります。また、マテリアライズドビューを作成するときにマテリアライズドビューごとに格納されるオプションの設定があります。ビューをリフレッシュしたり、ビューを最適化に使用したりするには、これらのオプション設定が現在のオプションと一致している必要があります。

部分的に処理を行った後にリフレッシュが失敗した場合、ビューは初期化されていない状態のままとなり、リフレッシュ開始前の状態にデータをリストアできません。リフレッシュが失敗したときに発生したエラーを調査して、この失敗の原因となった問題を解決し、REFRESH MATERIALIZED VIEW 文を再実行します。

また、ALTER MATERIALIZED VIEW 文の IMMEDIATE REFRESH 句を使用すると、基本となるデータが変更されたときすぐにビューを変更してリフレッシュできます。

接続に WITH HOLD 句を使用して開かれたカーソルがあり、このカーソルで文またはトランザクションのスナップショットを使用している場合、この文は実行できません。

パーミッション

マテリアライズドビューには INSERT パーミッションが必要です。マテリアライズドビュー定義のテーブルには SELECT パーミッションが必要です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

マテリアライズドビューを参照するすべてのオープンカーソルが閉じられます。

チェックポイントは実行の開始時に実行されます。

実行の開始時と終了時に自動コミットが実行されます。

実行中は、接続の blocking オプションを使用してリフレッシュされるマテリアライズドビューに排他スキーマロックがかけれます。また、マテリアライズドビューから参照されるすべてのテーブルにブロックなしで共有スキーマロックがかけられます。WITH 句を指定した場合は、基本となるテーブルに追加のロックを取得することがあります。さらに、リフレッシュが完了するまでマテリアライズドビューは初期化されていない状態になるため、データベースサーバーでクエリ最適化またはクエリ実行に使用できなくなります。

スナップショットアイソレーションを使用して REFRESH MATERIALIZED VIEW 文を実行した場合、データベースのトランザクションログには REFRESH 文のテキストと、マテリアライズドビューに挿入される個々のローのすべてのコピーが含まれます。データベースにリカバリが必要な場合、リカバリ後のビューの内容が REFRESH MATERIALIZED VIEW 文が最初に完了したときのビューの内容と正確に一致するようにするには、個々のローが必要です。さらに、データベースがミラーリングされている場合、トランザクションログ内の個々のローは個別に適用されます。そのため、スナップショットアイソレーションを使用した REFRESH MATERIALIZED VIEW 文の頻度を制限したり、BACKUP 文を使用してトランザクションログを定期的にトランケートしたりして、トランザクションログに必要なディスク領域の量を削減することが必要になる場合があります。

参照

- 「マテリアライズドビュー」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE MATERIALIZED VIEW 文」576 ページ
- 「トランザクションと独立性レベル」『SQL Anywhere サーバー SQL の使用法』
- 「独立性レベルと一貫性」『SQL Anywhere サーバー SQL の使用法』
- 「ALTER MATERIALIZED VIEW 文」427 ページ
- 「独立性レベルと一貫性」『SQL Anywhere サーバー SQL の使用法』
- 「blocking オプション」『SQL Anywhere サーバー データベース管理』
- 「テーブルロック」『SQL Anywhere サーバー SQL の使用法』
- 「スキーマロック」『SQL Anywhere サーバー SQL の使用法』
- 「BACKUP 文」489 ページ
- 「allow_snapshot_isolation オプション」『SQL Anywhere サーバー データベース管理』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューの制限」『SQL Anywhere サーバー SQL の使用法』
- 「sa_refresh_materialized_views システムプロシージャー」1158 ページ
- 「sa_materialized_view_info システムプロシージャー」1118 ページ
- 「sa_materialized_view_can_be_immediate システムプロシージャー」1116 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

EmployeeConfid99 というマテリアライズドビューを作成し、データを移植するとします。この処理は、次の文で実行できます。

```
CREATE MATERIALIZED VIEW EmployeeConfid99 AS
  SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary, ManagerID,
    Departments.DepartmentName, Departments.DepartmentHeadID
  FROM Employees, Departments
  WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid99;
```

後に、ビューが使用されるようになってから、READ COMMITTED 独立性レベル (独立性レベル 1) を使用してビューをリフレッシュし、ビューを再構築するとします。この処理は、次の文で実行できます。

```
REFRESH MATERIALIZED VIEW EmployeeConfid99
  WITH ISOLATION LEVEL READ COMMITTED
  FORCE BUILD;
```

警告
この例を実行し終わったら、作成したマテリアライズドビューを削除してください。このようにしないと、他の例を試すとき、基本となるテーブル Employees と Departments に対するスキーマ変更を実行できなくなります。有効化されている従属マテリアライズドビューを持つテーブルのスキーマは変更できません。「マテリアライズドビューの削除」『SQL Anywhere サーバー SQL の使用法』を参照してください。

REFRESH TEXT INDEX 文

テキストインデックスをリフレッシュします。

構文

REFRESH TEXT INDEX *text-index-name* **ON** [*owner.*]*table-name*
[**WITH** {
  ISOLATION LEVEL *isolation-level*
  | **EXCLUSIVE MODE**
  | **SHARE MODE** }]
[**FORCE** { **BUILD** | **INCREMENTAL** }]

パラメーター

WITH 句　WITH 句は、基本となるベーステーブルでリフレッシュ中に取得するロックの種類を指定する場合に使用します。取得したロックの種類によって、テキストインデックスの移植方法とトランザクションの同時実行性への影響が決まります。WITH 句を指定しなかった場合、デフォルトは、接続に設定されている独立性レベルに関係なく、WITH ISOLATION LEVEL READ UNCOMMITTED になります。

次の WITH 句オプションを指定できます。

○ **ISOLATION LEVEL *isolation-level***　WITH ISOLATION LEVEL は、リフレッシュ操作を実行する場合の独立性レベルを変更する場合に使用します。

接続の元の独立性レベルは、文の実行終了時にリストアされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **EXCLUSIVE MODE**　WITH EXCLUSIVE MODE は、独立性レベルを変更しないが、基本となるテーブルにコミットされたデータと矛盾しないようにデータを確実に更新する場合に使用します。WITH EXCLUSIVE MODE を使用すると、基本となるベーステーブルに排他テーブルロックがかけられます。リフレッシュ操作が完了するまで、他のトランザクションによって、基本となるテーブルに対する問い合わせ、更新、その他のアクションは実行できなくなります。テーブルのロックを取得できない場合、更新操作は失敗し、エラーが返されます。

○ **SHARE MODE**　WITH SHARE MODE は、リフレッシュ操作の実行中に、基本となるテーブルを他のトランザクションで読み込めるようにするために使用します。この句を指定すると、リフレッシュ操作が実行される前に基本となるベーステーブルの共有テーブルロックが取得され、リフレッシュ操作が完了するまで保持されます。

FORCE 句　この句は、リフレッシュ方法を指定する場合に使用します。この句を指定しなかった場合、データベースサーバーは、変更されたテーブルの割合に基づいて、増分更新するか、または完全再構築するかどうかを決定します。

○ **FORCE BUILD 句**　テキストインデックスを再作成することでリフレッシュします。この句は、テキストインデックスを強制的に完全再構築する場合に使用します。

○ **FORCE INCREMENTAL 句**　基本となるテーブルでの変更内容にのみ基づいてテキストインデックスをリフレッシュします。基本となるテーブルに加えられた更新の量が多くない場合、増分リフレッシュは短期間で完了します。この句は、テキストインデックスを強制的に増分更新する場合に使用します。

増分リフレッシュでは、削除済みのエントリはテキストインデックスから削除されません。このため、テキストインデックスのサイズは、現在のデータと履歴データを含めた予想サイズよりも大きくなる場合があります。通常、この問題は、常に FORCE INCREMENTAL 句を使用して手動でリフレッシュしているテキストインデックスで発生します。自動的にリフレッシュされるテキストインデックスでは、履歴データは、テキストインデックスの合計サイズの 50 % に達すると、自動的に削除されます。

備考

この文は、MANUAL REFRESH または AUTO REFRESH と定義されているテキストインデックスでのみ使用できます。

FORCE 句を使用した場合は、sa_text_index_stats システムプロシージャーの結果を確認して、完全再構築 (FORCE BUILD) または増分更新 (FORCE INCREMENTAL) のどちらが最適かを決定できます。

IMMEDIATE REFRESH と定義されているテキストインデックスに対して REFRESH TEXT INDEX 文を実行することはできません。

MANUAL REFRESH テキストインデックスの場合は、テキストインデックスを再表示するかどうかを、sa_text_index_stats システムプロシージャーを使用して判断します。pending_length を doc_length で除算して求められるパーセンテージを、再表示が必要かどうかを判断するうえでの目安にします。必要な再構築の種類を判断する場合も、deleted_length と doc_count を使って同じ処理を行います。

パーミッション

基本となるテーブルの所有者であるか、DBA 権限または REFERENCES パーミッションが必要です。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、この文は実行できません。

関連する動作

オートコミット。

参照

- 「テキストインデックスの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「トランザクションと独立性レベル」『SQL Anywhere サーバー SQL の使用法』
- 「独立性レベルと一貫性」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE TEXT INDEX 文」667 ページ
- 「ALTER TEXT INDEX 文」480 ページ
- 「DROP TEXT INDEX 文」733 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「テーブルロック」『SQL Anywhere サーバー SQL の使用法』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』
- 「テキストインデックスの再表示タイプ」『SQL Anywhere サーバー SQL の使用法』
- 「sa_refresh_text_indexes システムプロシージャー」1158 ページ
- 「sa_text_index_stats システムプロシージャー」1201 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、テキストインデックス MarketingTextIndex をリフレッシュし、強制的に再構築します。

```
REFRESH TEXT INDEX MarketingTextIndex ON MarketingInformation
  FORCE BUILD;
```

REFRESH TRACING LEVEL 文

トレーシングセッションが進行中に、sa_diagnostic_tracing_level テーブルからトレーシングレベルをリロードします。

構文

REFRESH TRACING LEVEL

備考

この文は、sa_diagnostic_tracing_level テーブルからトレーシングレベル情報をリロードするときに使用します。この文は、プロファイル対象のデータベースから呼び出す必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

トレーシングセッションを最初に開始すると、sa_diagnostic_tracing_level テーブルのローはサーバーメモリにロードされ、トレースする情報の種類を制御します。トレーシングセッションの停止と再起動を行うことなくトレースするデータの種類を変更するには、sa_diagnostic_tracing_level テーブルでローを手動で削除または挿入します。次に、REFRESH TRACING LEVEL 文を実行して設定をリロードします。

現在のトレーシングレベルを確認するには、次のように sa_diagnostic_tracing_level テーブルに問い合わせます。

```
SELECT * FROM sa_diagnostic_tracing_level WHERE enabled = 1;
```

パーミッション

DBA 権限

関連する動作

なし

参照

- 「ATTACH TRACING 文」487 ページ
- 「DETACH TRACING 文」706 ページ
- 「診断トレーシング」『SQL Anywhere サーバー SQL の使用法』
- 「sa_diagnostic_tracing_level テーブル」1022 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

パフォーマンス問題の解決に取り組んでいるとします。データベース全体のトレーシングレベルを高く設定し、問題の原因となるクエリを取得します。トレーシングセッションの開始後、システムの全ユーザーについて全クエリを取得するとデータベースの処理速度が大幅に遅くなることがわかりました。そのため、トレーシングを 1 ユーザーに制限し、そのユーザーからレポートされる問題を待機することにしました。ただし、設定を変更するときにトレーシングセッションを停止するつもりはありません。

Sybase Central ではデータベーストレーシングウィザードを使用してこの処理を実行できます。この方法をおすすめします。ただし、コマンドラインから実行することもできます。この場合、scope=DATABASE で enabled=1 の sa_diagnostic_tracing_level テーブルのローを、scope=USER、identifier=*userid*、enabled=1 などという同等のローで置換します。次に REFRESH TRACING LEVEL 文を実行し、新規の設定を使用してトレーシングを継続します。

RELEASE SAVEPOINT 文

現在のトランザクション内のセーブポイントを解放します。

構文

RELEASE SAVEPOINT [*savepoint-name*]

備考

セーブポイントを解放します。*savepoint-name* は、現在のトランザクションの SAVEPOINT 文で指定された識別子です。*savepoint-name* を省略すると、最新のセーブポイントが解放されます。

セーブポイントを解放しても、いずれの種類の COMMIT も実行されません。現在アクティブなセーブポイントのリストから、セーブポイントを削除するだけです。

パーミッション

現在のトランザクションに、対応する SAVEPOINT を入れておいてください。

関連する動作

なし

参照

- 「BEGIN TRANSACTION 文 [T-SQL]」500 ページ
- 「COMMIT 文」514 ページ
- 「ROLLBACK 文」890 ページ
- 「ROLLBACK TO SAVEPOINT 文」891 ページ
- 「SAVEPOINT 文」895 ページ
- 「トランザクション内のセーブポイント」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** RELEASE SAVEPOINT は、オプションの SQL/2008 言語機能 T271、"Savepoints" の一部です。

REMOTE RESET 文 [SQL Remote]

カスタムデータベース抽出プロシージャーで、単一トランザクションでの 1 リモートユーザーのすべてのサブスクリプションを起動します。

構文

REMOTE RESET *userid*

備考

この文は、1 人のリモートユーザーのすべてのサブスクリプションを 1 つのトランザクションで起動します。ISYSREMOTEUSER テーブルの log_sent 値と confirm_sent 値をトランザクションログの現在の位置に設定します。また、ISYSSUBSCRIPTION で作成し、起動した値を、このリモートユーザーのすべてのサブスクリプションに使用するトランザクションログの現在の位置に設定します。この文はコミットを実行しません。この呼び出しの後、明示的にコミットを実行してください。

ライブデータベース上で安全な抽出処理を書き込むには、サブスクリプションが開始されたのと同じトランザクション内で、データを独立性レベル 3 で抽出します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

この文を START SUBSCRIPTION の代わりに使用できます。START SUBSCRIPTION は、関連動作として暗黙的にコミットを実行するため、リモートユーザーが複数のサブスクリプションを持つ場合に START SUBSCRIPTION を使用して 1 つのトランザクションですべてのサブスクリプションを起動することはできません。

パーミッション

DBA 権限

関連する動作

この文のオートコミットはありません。

参照

- 「START SUBSCRIPTION 文 [SQL Remote]」941 ページ
- 「ISYSREMOTEUSER システムテーブル」1003 ページ
- 「複数のリモートデータベースの作成」『SQL Remote』
- 「サブスクリプションの開始」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、リモートユーザー SamS のサブスクリプションをリセットします。

```
REMOTE RESET SamS;
COMMIT;
```

REMOVE EXTERNAL OBJECT 文

データベースから外部オブジェクトを削除します。

構文

REMOVE EXTERNAL OBJECT *object-name*

パラメーター

object-name　外部オブジェクトの名前。

備考

外部環境の詳細については、「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』を参照してください。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』
- 「ALTER EXTERNAL ENVIRONMENT 文」420 ページ
- 「INSTALL EXTERNAL OBJECT 文」807 ページ
- 「START EXTERNAL ENVIRONMENT 文」939 ページ
- 「STOP EXTERNAL ENVIRONMENT 文」947 ページ
- 「SYSEXTERNENV システムビュー」1265 ページ
- 「SYSEXTERNENVOBJECT システムビュー」1266 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

REMOVE JAVA 文

データベースからクラスまたは JAR ファイルを削除します。

構文

REMOVE JAVA *classes-to-remove*

classes-to-remove :
  CLASS *java-class-name*, ... | **JAR** *jar-name*, ...

パラメーター

CLASS 句　*java-class-name* パラメーターは、削除される 1 つ以上の Java クラスの名前です。現在のデータベースにインストールされているクラスを指定します。

JAR 句　*jar-name* は、一重引用符で囲まれた最大の長さが 255 の文字列値です。

それぞれの *jar-name* には、現在のデータベースの保持された jar の *jar-name* の名前を指定します。*jar-name* と同じであることは、SQL システムの文字列比較ルールで決定されます。

備考

データベースからクラスまたは jar ファイルを削除します。クラスまたは jar は事前にインストールしておきます。クラスを削除すると、それはカラムまたは変数型として使用できなくなります。

この文は、Windows Mobile ではサポートされません。

パーミッション

- DBA 権限

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、現在のデータベースから Demo という名前の Java クラスを削除します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
REMOVE JAVA CLASS Demo;
```

次の文は、現在のデータベースから myJar という名前の Java jar を削除します。

```
REMOVE JAVA JAR 'myJar';
```

REORGANIZE TABLE 文

データベースへの連続アクセスという要件があるために、データベース全体の再構築ができない場合に、テーブルの断片化を解除します。

構文

REORGANIZE TABLE [*owner*.]*table-name*
[{ **PRIMARY KEY**
| **FOREIGN KEY** *foreign-key-name*
| **INDEX** *index-name* }]

パラメーター

次のいずれかの値に従って、テーブルを再編成します。

PRIMARY KEY 句　テーブルのプライマリキーのインデックスを再編成します。

FOREIGN KEY 句　指定の外部キーを再編成します。

INDEX 句　指定のインデックスを再編成します。

備考

テーブルが断片化されると、パフォーマンスが妨害されることがあります。この文を使って、テーブル内のローの断片化を解除したり、DELETE によって散在したインデックスを圧縮したりします。また、テーブルとそのインデックスを記録するために使われる総ページ数と、インデックスツリーに含まれるレベル数も減らします。ただし、データベースファイル全体のサイズは小さくなりません。sa_table_fragmentation と sa_index_density の各システムプロシージャーを使用して、テーブル対象の処理を選択することをおすすめします。

インデックスまたはキーを指定しない場合は、再編成処理によってローグループが削除されてから再挿入され、テーブル内のローの断片化が解除されます。グループごとに、テーブルの排他ロックが取得されます。グループの処理が完了すると、他の接続がテーブルにアクセスできるように、ロックが解除され、再取得されます (必要な場合は待機します)。グループの処理中はチェックポイントが中断されます。グループが終了すると、チェックポイントが発生することがあります。クラスタードインデックスが存在する場合、ローはクラスタードインデックスの順序で処理されます。存在しない場合は、プライマリキーの順序で処理されます。テーブルにクラスタードインデックスもプライマリキーもない場合、エラーが返されます。処理済みのローはテーブルの最後に再挿入され、処理の最後にローがプライマリキーによってクラスタされます。必要な作業量は、最初にローが断片化されていた程度に関係なく同じなので注意してください。

インデックスまたはキーを指定すると、そのインデックスが処理されます。オペレーション中は、テーブルの排他ロックが保持され、チェックポイントは中断されます。他の接続がテーブルにアクセスしようとすると、blocking オプションの設定に応じてブロックされるか失敗します。

ロック期間は、排他ロックを取得する前にインデックスページをあらかじめ読み込んでおくことで最短に抑えられます。

再編成で多数のページが修正される場合があるため、チェックポイントログが大きくなる可能性があります。これにより、データベースファイルのサイズが大きくなる可能性がありますが、チェックポイントログがシャットダウン時に削除され、ファイルはその時点でトランケートされるため、サイズは一時的に大きくなるだけです。

この文は、トランザクションログには記録されません。

文またはトランザクションのスナップショットを使用する、WITH HOLD 句を使用して開かれたカーソルがある場合、この文は実行できません。

この文の実行中に、進行状況メッセージの表示を要求できます。

また、Progress 接続プロパティを使用して、文がどの程度実行されたかを確認することもできます。

パーミッション

●テーブルの所有者であるか、DBA 権限が必要です。

●Windows Mobile ではサポートされません。

関連する動作

再編成を開始する前に、チェックポイントが実行され、空きページ数を最大限に増やそうとします。また、REORGANIZE TABLE 文の実行時は約 100 ローごとに暗黙的なコミットがあるため、大規模なテーブルを認識すると複数のコミットが実行されます。

標準と互換性

- **SQL/2008** ベンダー拡張。

参照

- Progress 接続プロパティ『SQL Anywhere サーバー データベース管理』
- 「progress_messages オプション」『SQL Anywhere サーバー データベース管理』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、Employees テーブルのプライマリキーのインデックスを再編成します。

```
REORGANIZE TABLE Employees
PRIMARY KEY;
```

次の文は、Employees テーブルのテーブルページを再編成します。

```
REORGANIZE TABLE Employees;
```

次の文は、Products テーブルのインデックス IX_prod_name を再編成します。

```
REORGANIZE TABLE Products
  INDEX IX_product_name;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

次の文は、Employees テーブルの外部キー FK_DepartmentID_DepartmentID を再編成します。

```
REORGANIZE TABLE Employees
  FOREIGN KEY FK_DepartmentID_DepartmentID;
```

RESIGNAL 文 [SP]

例外条件を送り返します。

構文

RESIGNAL [*exception-name*]

備考

例外ハンドラーの中で RESIGNAL を使って、まだアクティブな例外を持つ複合文を終了するか、別の指定された例外のレポートを終了できます。例外は、別の例外ハンドラーによって処理されるか、またはアプリケーションに返されます。

パーミッション

なし

関連する動作

なし

参照

- 「SIGNAL 文 [SP]」934 ページ
- 「BEGIN 文」497 ページ
- 「例外ハンドラー」『SQL Anywhere サーバー SQL の使用法』
- 「RAISERROR 文」862 ページ

標準と互換性

- **SQL/2008**　RESIGNAL 文は、オプションの SQL/2008 言語機能 P002、"Computational completeness" の一部です。
- **Transact-SQL**　RESIGNAL 文は、Transact-SQL 複合文およびプロシージャーでは使用できません。

例

次のフラグメントは SQLSTATE 52003 を除くすべての例外をアプリケーションに返します。

```
...
DECLARE COLUMN_NOT_FOUND EXCEPTION
  FOR SQLSTATE '52003';
...
EXCEPTION
WHEN COLUMN_NOT_FOUND THEN
SET message='Column not found';
WHEN OTHERS THEN
RESIGNAL;
```

RESTORE DATABASE 文

バックアップされたデータベースをアーカイブからリストアします。

構文

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*

パラメーター

CATALOG ONLY 句　指定されたアーカイブに関する情報を取り出し、それをバックアップ履歴ファイル (*backup.syb*) に保存します。アーカイブからデータをリストアするわけではありません。

RENAME 句　DB 領域ごとに新しいロケーションを指定できます。RENAME 句を DB 領域名の変更に使用することはできません。ただし、ファイル名の変更には RENAME 句を使用できます。

HISTORY 句　RESTORE DATABASE 操作を履歴ファイル *backup.syb* に記録するかどうかを制御できます。

KEY 句　空きページの削除をオンにした状態でバックアップされ、強力に暗号化されているアーカイブ済みデータベースを、リストアするための暗号化キーを指定できます。空きページの削除をオフにした状態でバックアップされている場合、データベースをリストアするための暗号化キーを指定する必要はありません。

バージョン 12 では、バージョン 11 以前のデータベースサーバーで作成したアーカイブバックアップをリストアできません。

備考

HISTORY OFF を指定しないかぎり、RESTORE DATABASE 操作を実行するたびにバックアップ履歴ファイル *backup.syb* が更新されます。backup.syb ファイルには、特定のデータベースサーバーで実行された BACKUP 操作と RESTORE 操作が記録されます。次の条件にあてはまる場合は、RESTORE DATABASE 操作が *backup.syb* に記録されないことがあります。

●RESTORE DATABASE 操作が頻繁に発生する

●*backup.syb* ファイルを定期的にアーカイブまたは削除するプロシージャーがない

●ディスク領域が非常に限られている

RESTORE DATABASE はリストアするデータベースを置き換えます。インクリメンタルバックアップが必要な場合は、BACKUP コマンドのイメージフォーマットを使用し、トランザクショ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ンログだけを保存してください。ただし、テープへのイメージバックアップはサポートされていません。

この文の実行中に、進行状況メッセージの表示を要求できます。

また、Progress 接続プロパティを使用して、文がどの程度実行されたかを確認することもできます。

パーミッション

この文を実行するのに必要なパーミッションは、サーバーコマンドラインで -gu オプションを使用して設定します。デフォルトの設定では、DBA 権限を必要とします。

この文は、Windows Mobile ではサポートされません。

関連する動作

なし

参照

- 「事前定義の DB 領域」『SQL Anywhere サーバー データベース管理』
- 「BACKUP 文」489 ページ
- 「バックアップとデータリカバリ」『SQL Anywhere サーバー データベース管理』
- 「SALOGDIR 環境変数」『SQL Anywhere サーバー データベース管理』
- FREE PAGE ELIMINATION 句、BACKUP 文 493 ページ
- 「-gu dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- Progress 接続プロパティ『SQL Anywhere サーバー データベース管理』
- 「progress_messages オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。
- **Windows Mobile**　Windows Mobile ではサポートされません。

例

次の例は、テープドライブからデータベースをリストアします。必要な円記号の数は、RESTORE DATABASE の実行時に接続しているデータベースに応じて異なります。データベースは、escape_character オプションの設定に影響します。通常は On に設定されていますが、utility_db では Off に設定されます。utility_db 以外のデータベースに接続している場合は、追加の円記号が必要です。

```
RESTORE DATABASE 'd:¥¥dbhome¥¥mydatabase.db'
FROM '¥¥¥¥.¥¥tape0';
```

RESUME 文

結果セットを返すカーソルの実行を再開します。

構文

RESUME *cursor-name*

cursor-name : *identifier* | *hostvar*

備考

この文は、結果セットを返すプロシージャーの実行を再開します。プロシージャーは、次の結果セット (INTO 句のない SELECT 文) が見つかるまで実行されます。プロシージャーが完了しても結果セットが見つからない場合は、SQLSTATE_PROCEDURE_COMPLETE 警告が設定されます。この警告は、SELECT 文に対してカーソルを RESUME するときにも設定されます。

RESUME 文は、Interactive SQL でサポートされていません。Interactive SQL で複数の結果セットを表示する場合、isql_show_multiple_result_sets オプションを ON に設定できます。または、[ツール] - [オプション] を選択して、[結果] タブの [複数の結果セットを表示] を選択します。

パーミッション

事前にカーソルを開いておきます。

関連する動作

なし

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「FETCH 文 [ESQL] [SP]」749 ページ
- 「結果セット」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次に Embedded SQL の例を示します。

```
1. EXEC SQL RESUME cur_employee;
2. EXEC SQL RESUME :cursor_var;
```

RETURN 文

関数、プロシージャー、またはバッチを無条件で終了し、オプションで値を返します。

構文

RETURN [*expression*]

備考

RETURN 文を使うと、SQL のブロックをすぐに終了できます。*expression* を指定する場合、*expression* の値を関数またはプロシージャーの値として返します。サブクエリは、*expression* では使用できません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

RETURN が内側の BEGIN ブロックにある場合、外側の BEGIN ブロックが終了します。

RETURN 文の後の文は実行されません。

関数の中では、式を関数の RETURNS データ型と同じデータ型にしてください。

プロシージャー内では、RETURN は Transact-SQL との互換性を確保するためのもので、整数のエラーコードを返すために使用されます。

パーミッション

なし

関連する動作

なし

参照

- 「CREATE FUNCTION 文」563 ページ
- 「CREATE FUNCTION 文 [外部呼び出し]」550 ページ
- 「CREATE FUNCTION 文 [Web サービス]」555 ページ
- 「CREATE PROCEDURE 文」602 ページ
- 「CREATE PROCEDURE 文 [外部呼び出し]」583 ページ
- 「CREATE PROCEDURE 文 [Web サービス]」591 ページ
- 「BEGIN 文」497 ページ
- 「RETURN 文を使って値を返す」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　コア機能。

例

次の関数は、3 つの数値の積を返します。

```
CREATE FUNCTION product (
  a NUMERIC,
  b NUMERIC,
  c NUMERIC )
RETURNS NUMERIC
BEGIN
  RETURN ( a * b * c );
END;
```

3 つの数値の積を計算します。

```
SELECT product(2, 3, 4);
```

product(2, 3, 4)
24

次のプロシージャーは、RETURN 文を使って、意味のない複雑なクエリを実行するのを避けます。

```
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  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;
```

REVOKE CONSOLIDATE 文 [SQL Remote]

このデータベースからの SQL Remote メッセージを統合データベースが受信するのを中止させます。

構文

REVOKE CONSOLIDATE FROM *userid*

備考

統合データベースを表すユーザー ID に対して CONSOLIDATE パーミッションをリモートデータベースで付与します。REVOKE CONSOLIDATE 文は、現在のデータベースからメッセージを受信するユーザーリストから統合データベースのユーザー ID を削除します。

パーミッション

DBA 権限

関連する動作

オートコミット。指定したユーザーのすべてのサブスクリプションを削除します。

参照

- 「REVOKE PUBLISH 文 [SQL Remote]」885 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」887 ページ
- 「REVOKE REMOTE DBA 文 [SQL Remote]」886 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」776 ページ
- 「CONSOLIDATE パーミッションの取り消し」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

●次の文は、ユーザー ID condb の統合ステータスを取り消します。

```
REVOKE CONSOLIDATE FROM condb;
```

REVOKE PUBLISH 文 [SQL Remote]

名前を指定したユーザー ID を現在のパブリッシャーとして識別するのを中止します。

構文

REVOKE PUBLISH FROM *userid*

備考

SQL Remote インストール環境にある各データベースは、出力メッセージ内でパブリッシャーのユーザー ID によって識別されます。現在のパブリッシャーのユーザー ID は、特別値 CURRENT PUBLISHER を使用して検索できます。次のクエリは、現在のパブリッシャーを識別します。

```
SELECT CURRENT PUBLISHER;
```

REVOKE PUBLISH 文により、名前を指定したユーザー ID がパブリッシャーとして識別されなくなります。

データベースにアクティブな SQL Remote パブリケーションまたはサブスクリプションがあるときは、そのデータベースから REVOKE PUBLISH を実行しないでください。

データベースで REVOKE PUBLISH 文を実行すると、SQL Remote インストール環境に次のような影響を及ぼします。

●CURRENT PUBLISHER カラムがプライマリキーの一部になっているテーブルにデータを挿入できなくなります。また、出力メッセージがパブリッシャーユーザー ID で識別できなくなり、受信側データベースによって受け取られなくなります。

SQL Remote インストール環境の統合データベースまたはリモートデータベースでパブリッシャーユーザー ID を変更する場合はそのデータベースからメッセージを受信するすべてのデータベースで、新しいパブリッシャーユーザー ID に REMOTE パーミッションが付与されていることを確認します。通常この作業を行うには、すべてのサブスクリプションを一度削除し、再作成する必要があります。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「GRANT PUBLISH 文 [SQL Remote]」777 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」887 ページ
- 「REVOKE REMOTE DBA 文 [SQL Remote]」886 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」884 ページ
- 「CURRENT PUBLISHER 特別値」62 ページ
- 「PUBLISH パーミッションの取り消し」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

```
REVOKE PUBLISH FROM publisher_ID;
```

REVOKE REMOTE DBA 文 [SQL Remote]

ユーザー ID から REMOTE DBA 権限を取り消します。

構文 1

REVOKE REMOTE DBA
FROM *userid*, ...

備考

Mobile Link では、REMOTE DBA 権限は SQL Anywhere 同期クライアント (dbmlsync) で必要なパーミッションのレベルを指します。

SQL Remote では、REMOTE DBA 権限は、DBA ユーザー ID パスワードを送信する場合のセキュリティを確保しながら、SQL Remote Message Agent がデータベースにフルアクセスしてメッセージ内での変更操作をできるようにするものです。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「REVOKE PUBLISH 文 [SQL Remote]」885 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」887 ページ
- 「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」778 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」884 ページ
- 「同期の開始」『Mobile Link クライアント管理』
- 「REMOTE DBA 権限」『SQL Remote』
- 「REMOTE DBA 権限の取り消し」『SQL Remote』

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、ユーザー S_Beaulieu から REMOTE DBA パーミッションを取り消します。

```
REVOKE REMOTE DBA FROM S_Beaulieu;
```

REVOKE REMOTE 文 [SQL Remote]

このデータベースからの SQL Remote メッセージをユーザーが受信できないようにします。

構文

REVOKE REMOTE FROM *userid*, ...

備考

ユーザー ID が SQL Remote レプリケーションインストール環境でメッセージを受信するには、REMOTE パーミッションが必須です。REVOKE REMOTE 文は、現在のデータベースからメッセージを受信するユーザーリストからユーザー ID を削除します。

パーミッション

DBA 権限

関連する動作

オートコミット。指定したユーザーのすべてのサブスクリプションを削除します。

参照

- 「REVOKE PUBLISH 文 [SQL Remote]」885 ページ
- 「GRANT REMOTE 文 [SQL Remote]」780 ページ
- 「REVOKE REMOTE DBA 文 [SQL Remote]」886 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」884 ページ
- 「REMOTE パーミッションの取り消し」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

```
REVOKE REMOTE FROM SamS;
```

REVOKE 文

ユーザーからパーミッションを削除します。

構文 1 - 権限の取り消し

REVOKE *permission*, … **FROM** *userid*, …

permission :
BACKUP
| **CONNECT**
| **CREATE ON** *dbspace*
| **DBA**
| **GROUP**
| **INTEGRATED LOGIN**
| **KERBEROS LOGIN**
| **MEMBERSHIP IN GROUP** *userid*, …
| **PROFILE**
| **READCLIENTFILE**
| **READFILE**
| **RESOURCE**
| **VALIDATE**
| **WRITECLIENTFILE**

構文 2 - データベースオブジェクトパーミッションの取り消し

REVOKE *table-permission*, …
ON [*owner*.]*table-name*
FROM *userid*, …

table-permission :
ALL [**PRIVILEGES**]
| **ALTER**
| **DELETE**
| **INSERT**
| **REFERENCES** [**(** *column-name*, … **)**]
| **SELECT** [**(** *column-name*, … **)**]
| **UPDATE** [**(** *column-name*, … **)**]

構文 3

REVOKE EXECUTE
ON [*owner*.]*procedure-name*
FROM *userid*, …

構文 4

REVOKE USAGE ON SEQUENCE *sequence-name*
FROM *userid*, …

備考

REVOKE 文は、GRANT 文を使って付与したパーミッションを削除します。構文 1 は、特殊ユーザーパーミッションを取り消します。構文 2 は、テーブルパーミッションを取り消します。構文 3 は、プロシージャーの EXECUTE パーミッションを取り消します。構文 4 は、シーケンスパーミッションを取り消します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

REVOKE CONNECT はデータベースからユーザー ID を削除すると一緒に、そのユーザーが所有しているオブジェクト (テーブル、ビュー、プロシージャーなど) と、そのユーザーによって付与されたパーミッションも破棄します。削除されているユーザーが、別のユーザーが所有するビューによって参照されるテーブルを所有する場合、そのユーザーに対して REVOKE CONNECT を実行することはできません。

REVOKE GROUP は、グループのすべてのメンバーから MEMBERSHIP IN GROUP を自動的に取り消します。

グループにユーザーを追加すると、ユーザーはそのグループに割り当てられたすべてのパーミッションを継承します。SQL Anywhere では、ユーザーがグループのメンバーとして継承するパーミッションのサブセットを取り消すことはできません。取り消すことができるのは、GRANT 文によって明示的に付与されるパーミッションだけです。異なるユーザーに別々のパーミッションを付与する必要がある場合、適切なパーミッションを持つグループを別々に作成するか、必要なパーミッションを各ユーザーに明示的に付与することができます。

テーブル、ビュー、プロシージャーに対するグループパーミッションを付与したり取り消したりすると、グループのメンバー全員がその変更を継承します。DBA、RESOURCE、GROUP の各パーミッションについては継承されません。これらのパーミッションは、必要に応じて各ユーザー ID に割り当てる必要があります。

ユーザーへのパーミッションを取り消し、さらにパーミッションに WITH GRANT OPTION があった場合、そのユーザーがパーミッションを付与していたすべてのユーザーへのパーミッションも取り消されます。たとえば、UserA SELECT...WITH GRANT OPTION パーミッションをテーブルと UserA に付与し、UserB にそのテーブルの SELECT パーミッションを付与します。UserA から SELECT パーミッションを取り消すと、UserB でも取り消されます。

REVOKE USAGE ON SEQUENCE は、シーケンス内の現在の値または次の値を評価するためのユーザーのパーミッションを削除します。この文を実行するには、DBA 権限が必要です。または、シーケンスの作成者である必要があります。

パーミッション

取り消されるパーミッションの付与者であるか、DBA 権限が必要です。

接続パーミッションまたは別のユーザーからのテーブルパーミッションを取り消す場合、他のユーザーはそのデータベースに接続できません。DBO からの接続パーミッションの取り消しはできません。

ユーティリティデータベースに接続するときに REVOKE CONNECT FROM DBA を実行すると、以降のユーティリティデータベースへの接続が無効になります。これは、REVOKE CONNECT を実行する前に確立していた接続を使用するか、データベースサーバーを再起動しないかぎり、以降はユーティリティデータベースに接続できないことを意味します。

関連する動作

オートコミット。

参照

- 「GRANT 文」782 ページ

標準と互換性

- **SQL/2008**　構文 1 はベンダー拡張です。構文 2 と構文 3 は、SQL/2008 標準のコア機能です。構文 2 を使用する場合、SQL Anywhere では PRIVILEGES キーワードは任意ですが、SQL/2008 標準では "ALL PRIVILEGES" は必須です。

  構文 4 は、オプションの SQL/2008 言語機能 T176、"Sequence generator support" の一部です。

例

ユーザー Dave が Employees テーブルを更新できないようにします。

```
REVOKE UPDATE ON Employees FROM Dave;
```

ユーザー Jim からリソースパーミッションを取り消します。

```
REVOKE RESOURCE FROM Jim;
```

ユーザープロファイル名 Administrator から統合化ログインマッピングを取り消します。

```
REVOKE INTEGRATED LOGIN FROM Administrator;
```

Finance グループがプロシージャー ShowCustomers を実行できないようにします。

```
REVOKE EXECUTE ON ShowCustomers FROM Finance;
```

ユーザー ID FranW をデータベースから削除します。

```
REVOKE CONNECT FROM FranW;
```

ROLLBACK 文

トランザクションを終了し、最後に行った COMMIT または ROLLBACK 以降の変更を元に戻します。

構文

ROLLBACK [WORK]

備考

トランザクションとは、データベース接続においてデータベースに対して実行する作業の論理単位であり、COMMIT 文または ROLLBACK 文で囲まれた部分のことです。ROLLBACK 文は、現在のトランザクションを終了し、前回の COMMIT または ROLLBACK 以降にデータベースに対して行われたすべての変更を元に戻します。

Interactive SQL では、次の方法でも ROLLBACK を実行できます。

●[Ctrl + Shift + R] キーを押す。

●**[SQL]** » **[ロールバック]** を選択します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Interactive SQL では、**[SQL]** メニューまたはキーボードショートカットから ROLLBACK を実行しても、**[SQL 文]** ウィンドウ枠の内容は変更されません。ただし、**[結果]** ウィンドウ枠の **[結果]** タブはクリアされます。

パーミッション

なし

関連する動作

WITH HOLD 句で開かれた以外のすべてのカーソルを閉じます。

参照

- 「COMMIT 文」514 ページ
- 「Interactive SQL での COMMIT 文と ROLLBACK 文の実行」『SQL Anywhere サーバー データベース管理』
- 「ROLLBACK TO SAVEPOINT 文」891 ページ
- 「Interactive SQL オプション」『SQL Anywhere サーバー データベース管理』
- 「変更のキャンセル」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　コア機能。

ROLLBACK TO SAVEPOINT 文

SAVEPOINT の後に加えられた変更を取り消します。

構文

ROLLBACK TO SAVEPOINT [*savepoint-name*]

備考

ROLLBACK TO SAVEPOINT 文は、SAVEPOINT が作成されてから加えられた変更を取り消します。SAVEPOINT の前に加えられた変更は取り消されず、そのまま残ります。

savepoint-name は、現在のトランザクションの SAVEPOINT 文で指定された識別子です。*savepoint-name* を省略すると、直前のセーブポイントが使用されます。指定したセーブポイントの後にあるセーブポイントは自動的に解放されます。

パーミッション

現在のトランザクションに、対応する SAVEPOINT を入れておいてください。

関連する動作

なし

参照

- 「BEGIN TRANSACTION 文 [T-SQL]」500 ページ
- 「COMMIT 文」514 ページ
- 「RELEASE SAVEPOINT 文」873 ページ
- 「ROLLBACK 文」890 ページ
- 「SAVEPOINT 文」895 ページ
- 「トランザクション内のセーブポイント」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　ROLLBACK TO SAVEPOINT は、SQL/2008 標準のオプションの SQL 言語機能 T271 の一部です。

ROLLBACK TRANSACTION 文 [T-SQL]

SAVE TRANSACTION の後に加えられた変更を取り消します。

構文

ROLLBACK TRANSACTION [*savepoint-name*]

備考

ROLLBACK TRANSACTION 文は、SAVE TRANSACTION を使用してセーブポイントが作成されてから加えられた変更を取り消します。SAVE TRANSACTION の前に加えられた変更は取り消されず、そのまま残ります。

savepoint-name は、現在のトランザクションの SAVE TRANSACTION 文で指定された識別子です。*savepoint-name* を省略すると、未処理のまま残っている変更がすべてロールバックされます。指定したセーブポイントの後にあるセーブポイントは自動的に解放されます。

パーミッション

現在のトランザクションに、対応する SAVE TRANSACTION を入れておいてください。

関連する動作

なし

参照

- 「ROLLBACK TO SAVEPOINT 文」891 ページ
- 「BEGIN TRANSACTION 文 [T-SQL]」500 ページ
- 「COMMIT 文」514 ページ
- 「SAVE TRANSACTION 文 [T-SQL]」894 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例は、値 10、20 などの 5 つのローを表示します。DELETE の効果は ROLLBACK TRANSACTION 文によって取り消されますが、前の INSERT または UPDATE の効果は取り消されません。

```
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
```

ROLLBACK TRIGGER 文

トリガーによって加えられた変更を取り消します。

構文

ROLLBACK TRIGGER [**WITH** *raiserror-statement*]

備考

ROLLBACK TRIGGER 文は、トリガーを起動するデータの修正など、トリガーの中で実行された処理を元に戻します。

必要に応じて、RAISERROR 文を実行できます。RAISERROR 文を実行すると、エラーがアプリケーションに返されます。RAISERROR 文を実行しないと、エラーは返されません。

ネストしたトリガーの中で ROLLBACK TRIGGER 文を使用し、RAISERROR 文を発行しないと、最も内側のトリガーとそのトリガーを起動した文だけが元に戻されます。

パーミッション

なし

関連する動作

なし

参照

- 「CREATE TRIGGER 文」670 ページ
- 「ROLLBACK 文」890 ページ
- 「ROLLBACK TO SAVEPOINT 文」891 ページ
- 「RAISERROR 文」862 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

- **Transact-SQL**　ROLLBACK TRIGGER は、Watcom SQL と Transact-SQL の両方のストアドプロシージャーでサポートされています。ROLLBACK TRIGGER は Adaptive Server Enterprise でサポートされています。

SAVE TRANSACTION 文 [T-SQL]

現在のトランザクション内でセーブポイントを確立します。

構文

SAVE TRANSACTION *savepoint-name*

備考

現在のデータベース内でセーブポイントを確立します。*savepoint-name* は ROLLBACK TRANSACTION 文の中で使用できる識別子です。トランザクションが終了すると、すべてのセーブポイントは自動的に解放されます。

パーミッション

なし

関連する動作

なし

参照

- 「SAVEPOINT 文」895 ページ
- 「トランザクション内のセーブポイント」『SQL Anywhere サーバー SQL の使用法』
- 「BEGIN TRANSACTION 文 [T-SQL]」500 ページ
- 「COMMIT 文」514 ページ
- 「ROLLBACK TRANSACTION 文 [T-SQL]」892 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、値 10、20 などの 5 つのローを表示します。DELETE の効果は ROLLBACK TRANSACTION 文によって取り消されますが、前の INSERT または UPDATE の効果は取り消されません。

```
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
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SAVEPOINT 文

現在のトランザクション内でセーブポイントを確立します。

構文

SAVEPOINT [*savepoint-name*]

備考

現在のデータベース内でセーブポイントを確立します。*savepoint-name* は RELEASE SAVEPOINT または ROLLBACK TO SAVEPOINT 文の中で使用できる識別子です。トランザクションが終了すると、すべてのセーブポイントは自動的に解放されます。

トリガーまたはアトミックな複合文の実行中に確立されたセーブポイントは、アトミックオペレーションが終了すると自動的に解放されます。

セーブポイント内からプロキシテーブルのデータは修正できません。

パーミッション

なし

関連する動作

なし

参照

- 「RELEASE SAVEPOINT 文」873 ページ
- 「ROLLBACK TO SAVEPOINT 文」891 ページ
- 「SAVE TRANSACTION 文 [T-SQL]」894 ページ
- 「トランザクション内のセーブポイント」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　SAVEPOINT 文は、オプションの SQL/2008 言語機能 T271、"Savepoints" の一部です。
- **Transact-SQL**　Transact-SQL では、SAVE TRANSACTION 文を使用してセーブポイントを作成します。

SELECT 文

データベースから情報を取得します。

構文

[**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* }]

[**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*]
[*row-limitation-option-2*]
[**OPTION(** *query-hint*, ... **)**]

temporary-views :
regular-view, ...
| **RECURSIVE** { *regular-view* | *recursive-view* }, ...

regular-view :
view-name [**(** *column-name*, ... **)**]
AS (*subquery* **)**

recursive-view :
view-name **(** *column-name*, ... **)**
AS (*initial-subquery* **UNION ALL** *recursive-subquery* **)**

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**]
FROM *table-name*

sequence-expression : 「式」12 ページを参照してください。

from-expression : 「FROM 句」758 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

group-by-expression : 「GROUP BY 句」788 ページを参照してください。

search-condition : 「探索条件」34 ページを参照してください。

window-name : *identifier*

window-expression : 「WINDOW 句」989 ページを参照してください。

window-spec : 「WINDOW 句」989 ページを参照してください。

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**

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| **FORCE NO OPTIMIZATION**
| *option-name*=*option-value*

option-name : *identifier*

option-value :
hostvar (許可されたインジケーター)
| *string*
| *identifier*
| *number*

パラメーター

WITH 句または WITH RECURSIVE 句　1 つ以上の共通テーブル式を定義します。共通テーブル式はテンポラリビューでもあり、文の残りの部分に使用されます。これらの式は、非再帰的か自己再帰的のいずれかに指定できます。再帰的な共通テーブル式は、RECURSIVE キーワードが指定されている場合にのみ、単独で表示されるか、非再帰的なテーブル式と混合で表示されます。相互再帰的な共通テーブル式は、サポートされていません。

SELECT クエリブロックが次のいずれかの場所に表示されている場合のみ、この句を使用できます。

○ビュー定義の最上位レベルの SELECT クエリブロックを含む最上位レベルの SELECT クエリブロック内

○INSERT クエリブロック内の最上位レベルの SELECT 文内

○任意のタイプの SQL 文で派生テーブルを定義しているネストされた SELECT クエリブロック内

再帰的な式は、初期のサブクエリと再帰的なサブクエリから構成されます。初期クエリは、ビューのスキーマを暗黙的に定義します。再帰的なサブクエリには、FROM 句内のビューへの参照を入れてください。それぞれの反復中に、この参照によって前の反復でビューに追加されたローだけが参照されます。参照は、外部ジョインの NULL 入力側では表示できません。再帰的な共通テーブル式は、集合関数を使用できません。また、GROUP BY、ORDER BY、DISTINCT の各句を含むことはできません。

リモートテーブルでは WITH 句はサポートされません。WITH 句は、INTERSECT、UNION、EXCEPT クエリブロック内でも使用されることがあります。

ALL 句または DISTINCT 句　All (デフォルト) は、SELECT 文の句を満たすすべてのローを返します。DISTINCT を指定すると、重複した出力ローが削除されます。多くの場合、DISTINCT を指定すると、文の実行時間が非常に長くなります。したがって、DISTINCT を使用するのは、必要な場合だけにしてください。

***row-limitation* 句**　row-limitation 句を使用することによって、WHERE 句を満たすローのサブセットのみを返すことができます。*row-limitation* 句は、一度に 1 つのみ指定できます。これらの句を指定する場合は、ローの順序を意味のあるものにするために ORDER BY 句も指定する必要があります。

○ ***row-limitation-option-1***　TOP 引数と START AT 引数には、ホスト変数、整数定数、または整数変数を使用した簡単な算術演算を指定できます。TOP 引数は 0 以上にします。START AT 引数は 0 より大きい値にします。

startat-expression が指定されていない場合、デフォルトは 1 です。TOP の引数が ALL の場合、*startat-expression* で始まるすべてのローが返されます。TOP limit-expression START AT startat-expression 句は、LIMIT (startat-expression -1), limit-expression 、または LIMIT limit-expression OFFSET (startat-expression -1) と同等です。

○ ***row-limitation-option-2***　LIMIT 引数と OFFSET 引数には、ホスト変数、整数定数、または整数変数を使用した簡単な算術演算を指定できます。LIMIT 引数は 0 以上の値、OFFSET 引数は 0 以上の値に評価される必要があります。*offset-expression* が指定されていない場合は、デフォルトの 0 になります。

ロー制限句 LIMIT offset-expression, limit-expression は LIMIT limit-expression OFFSET offset-expression と同等です。どちらの構成も TOP limit-expression START AT (offset-expression + 1) と同等です。

LIMIT キーワードはデフォルトでは無効になっています。LIMIT キーワードを有効にするには、reserved_keywords オプションを使用します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

***select-list* 句**　*select-list* は、カンマで区切られた式のリストであり、データベースから何を取り出すかを指定します。アスタリスク (*) は、FROM 句に記述された全テーブルのすべてのカラムを選択することを意味します。

集合関数は、*select-list* で許可されています。サブクエリも、*select-list* で許可されます。それぞれのサブクエリは、カッコで囲みます。

エイリアス名はクエリを通じて使用でき、エイリアスの式を表します。

エイリアス名も、SELECT 文から出力された各カラムの最上部に、Interactive SQL で表示されます。オプションのエイリアス名を式の後で指定しないと、Interactive SQL は式を表示します。

INTO 句　次の 3 つの INTO 句を使用します。

○ **INTO *hostvar-list* 句**　この句は Embedded SQL でだけ使用されます。これは SELECT 文の結果が移動する場所を指定します。*select-list* 内のそれぞれの項目に対して、1 つのホスト変数項目が必要です。*select-list* 項目は、順番にホスト変数の中に置かれます。インジケーターホスト変数も各ホスト変数と一緒に使用でき、プログラムは *select-list* 項目が NULL であったかどうかを通知できます。

クエリの結果、選択されたローが存在しない場合、変数は更新されず、ローが見つからないことを示す警告が表示されます。

○ **INTO *variable-list* 句**　この句はプロシージャーとトリガーの中でだけ使用されます。これは SELECT 文の結果が移動する場所を指定します。*select-list* 内のそれぞれの項目に対して、1 つの変数が必要です。*select-list* 項目は、順番に変数の中に置かれます。

○ **INTO *table-name* 句**　この句は、テーブルの作成とデータの挿入に使用します。

永久テーブルを作成するには、クエリが次のいずれかの条件を満たしている必要があります。

●*select-list* に複数の項目が含まれ、INTO ターゲットが単一の *table-name* 識別子である。

●*select-list* に * が含まれ、INTO ターゲットが *owner.table* として指定されている。

1 つのカラムを持つ永久テーブルを作成するには、テーブル名を *owner.table* として指定します。

この文では、テーブルを作成する副作用として実行前に COMMIT が行われます。この文を実行するには RESOURCE 権限が必要です。新規テーブルにパーミッションは付与されません。この文は、CREATE TABLE とそれに続く INSERT...SELECT の省略形です。

この句を使用して作成されるテーブルには、プライマリキーは定義されていません。ALTER TABLE を使用してプライマリキーを追加できます。プライマリキーは、テーブルに UPDATE または DELETE を適用する前に追加してください。そうしないと、これらのオペレーションによって、影響を受けるローのトランザクションログにすべてのカラム値が記録されます。

INTO LOCAL TEMPORARY TABLE　この句は、ローカルテンポラリテーブルを作成し、クエリの結果を移植するために使用します。この句を使用する場合、テンポラリテーブル名の先頭に # を付ける必要はありません。

FROM 句　*table-expression* の中で指定されるテーブルとビューからローを取り出します。FROM 句を指定しない SELECT 文を使って、テーブルから取り出せなかった式の値を表示できます。たとえば、この 2 つの文は同じです。また、グローバル変数 @@version の値を表示します。

```
SELECT @@version;
SELECT @@version FROM DUMMY;
```

WHERE 句　この句は、FROM 句の中で指定したテーブルから選択するローを指定します。この句を FROM 句の一部である ON フレーズの代わりに使用すると、複数のテーブルをジョインできます。

GROUP BY 句　カラム、エイリアス名、または関数によってグループ分けできます。クエリの結果には、指定したカラム、エイリアス、または関数の中の個別の値の各セットに対し 1 つのローが入ります。DISTINCT や、UNION、INTERSECT、EXCEPT などの集合操作と同じように、GROUP BY 句は NULL 値を各ドメインの他の値と同様に扱います。つまり、グループ化属性に複数の NULL 値が存在すると 1 つのグループが形成されます。集合関数をこれらのグループに適用して、意味のある結果を取得することができます。

GROUP BY を使う場合、*select-list*、HAVING 句、ORDER BY 句が参照できるのは、GROUP BY 句の中で指定した識別子だけです。ただし、*select-list* と HAVING 句は集合関数を持つことができます。

HAVING 句　この句は、個々のロー値ではなくグループ値に基づいてローを選択します。HAVING 句を使用できるのは、文に GROUP BY 句があるか、*select-list* が集合関数のみから成る場合だけです。HAVING 句で参照されるカラム名は、GROUP BY 句に含まれるか、または HAVING 句の集合関数のパラメーターとして使用されます。

WINDOW 句　この句は、AVG や RANK などの Window 関数を使用するウィンドウのすべてまたは一部を定義します。

ORDER BY 句　この句はクエリの結果をソートします。ORDER BY リストの各項目には、昇順の場合 (デフォルト) は ASC、降順の場合は DESC のラベルを付けることができます。式が整数 *n* である場合、クエリの結果は *select-list* の *n* 番目の項目でソートされます。

特定の順序でローが返されるようにする唯一の方法は ORDER BY を使用することです。ORDER BY 句がない場合は、SQL Anywhere が最も効率のよい順序でローを返します。つまり、ローに最後にアクセスした日付やその他の要因によって、結果セットでの表示順序が異なることがあります。

Embedded SQL の場合は、データベースから結果を取得し、その値を INTO 句でホスト変数に格納するために、SELECT 文を使用します。SELECT 文は、1 つのローだけを返さなければなりません。複数のローを対象にクエリを実行する場合は、カーソルを使います。

FOR UPDATE または FOR READ ONLY 句　これらの句は、クエリに対して開かれているカーソル経由で更新を許可するかどうかを指定し、許可する場合、使用する同時実行性のセマンティックを指定します。この句は、FOR XML 句と一緒に使用できません。

SELECT 文の FOR 句を使用しない場合、カーソルの更新可能性は、カーソルの宣言とカーソルの同時実行性を API が指定する方法によって変わります。ODBC、JDBC、OLE DB、ADO.NET、

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Embedded SQL では、文の更新可能性は明示的になり、アプリケーションが上書きしなければ、読み込み専用カーソルが使用されます。Open Client とストアドプロシージャーでは、カーソルの更新可能性を指定する必要はありません。また、デフォルトは FOR UPDATE です。

Open Client とストアドプロシージャーでは、カーソルの更新可能性と文の更新可能性は ansi_update_constraints データベースオプションの設定、文の特徴に応じて変わります。たとえば、ORDER BY、DISTINCT、GROUP BY、HAVING、UNION、集合関数、ジョイン、非更新可能なビューなどです。ストアドプロシージャーでは、ORDER BY 句なしの単一テーブルのクエリの場合、または ansi_update_constraints オプションが Off に設定されている場合、カーソルのデフォルトは FOR UPDATE です。ansi_update_constraints オプションが Cursors または Strict に設定されている場合、ORDER BY 句を含むクエリ上のカーソルのデフォルトは READ ONLY です。ただし、FOR UPDATE 句を使用して、カーソルを更新可能と明示的に示すこともできます。ORDER BY 句またはジョインを使用してカーソルに対する更新を可能にするにはコストがかかるため、2 つ以上のテーブルのジョインを含むクエリに対するカーソルは READ ONLY であり、ansi_update_constraints データベースオプションが Off でないかぎり、更新可能にすることはできません。

FOR READ ONLY として宣言されたカーソルは、UPDATE (位置付け) 文、DELETE (位置付け) 文、または PUT 文には使用できません。FOR READ ONLY は Embedded SQL のデフォルトです。

FOR UPDATE 句は、明示的にカーソルを更新可能にします。FOR UPDATE を単独で使用しても、そのこと自体では、文の結果セットのローに対する同時実行性の制御には影響を及ぼしません。このことを行うには、FOR UPDATE BY LOCK または FOR UPDATE BY [VALUES | TIMESTAMP] のいずれかを指定します。

○ **FOR UPDATE BY LOCK 句**　データベースサーバーは、結果セットのフェッチされたローに対する意図的ローロックを取得します。このロックは長期間のロックであり、トランザクションがコミットまたはロールバックされるまで保持されます。

○ **FOR UPDATE BY { VALUES | TIMESTAMP }**　FOR UPDATE BY TIMESTAMP または FOR UPDATE BY VALUES を指定すると、データベースサーバーはキーセット駆動型の (value-sensitive) カーソルを使用して最適な同時実行性を使用します。このような状況では、アプリケーションが (別の文を使用して) カーソルの範囲外のローを変更した場合、または別の接続でローが変更されたことを示す警告やエラーがサーバーによって生成されてもアプリケーションがそれを無視する場合、更新内容が消失する可能性があります。

文で意図的なロックをかけるには、次のいずれかを実行します。

○クエリで FOR UPDATE BY LOCK を指定

○クエリの FROM 句で HOLDLOCK、WITH (HOLDLOCK)、WITH (UPDLOCK)、または WITH (XLOCK) を指定

○CONCUR_LOCK を指定する API の読み出しでカーソルを開く

○更新のフェッチを示す属性でローをフェッチ

FOR UPDATE OF 句は、UPDATE (位置付け) 文、DELETE (位置付け) 文、または PUT 文で変更可能なカラムに明示的に名前を付けます。この句は、他の FOR UPDATE 句、FOR READ ONLY 句、または FOR XML 句と一緒には使用できません。

○ **FOR UPDATE OF *column-list* 句**　FOR UPDATE OF 句を指定すると、データベースサーバーは、位置付け UPDATE 文または位置付け DELETE 文で変更可能なカラムを、この句で明示的に指定されたカラムに制限します。別のカラムを変更しようとすると、カラムが見つからないというエラーになります。リスト内で参照されているカラムが実際に存在するかどうか、および、そのカラムのテーブルが更新可能なテーブルであるかどうかを調べるチェックは行われません。

FOR XML 句　この句は、結果セットが XML ドキュメントとして返されるように指定します。XML のフォーマットは、指定するモードによって異なります。この句は、FOR UPDATE 句または FOR READ ONLY 句と一緒には使用できません。FOR XML で宣言したカーソルは暗黙的に READ ONLY になります。

RAW モードを指定すると、結果セットの各ローは XML <row> 要素として表され、各カラムは <row> 要素の属性として表されます。

AUTO モードを指定すると、クエリの結果は、ネストされた XML 要素として返されます。*select-list* 内で参照される各テーブルは、XML 内で要素として表されます。要素のネスト順は、テーブルが *select-list* 内で参照される順序に基づいています。

EXPLICIT モードを指定すると、生成された XML ドキュメントの形式を制御できます。EXPLICIT モードにすると、RAW モードや AUTO モードに比べて、要素の名前付けとネスト構造の指定がより柔軟にできます。

OPTION 句　この句は、クエリの処理方法についてヒントを示します。次のクエリヒントがサポートされます。

○ **MATERIALIZED VIEW OPTIMIZATION 句**　MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザーがクエリを処理するときにマテリアライズドビューを使用する方法を指定します。指定した *option-value* は、このクエリでのみ materialized_view_optimization データベースオプションを上書きします。*option-value* の可能な値は、materialized_view_optimization database オプションに使用できる値と同じです。

○ **FORCE OPTIMIZATION 句**　クエリ指定に単純なクエリしか含まれない場合 (特定の行を識別する WHERE 句に等号条件を含んだ単一ブロック、単一テーブルのクエリ)、通常、処理中にコストベースの最適化はバイパスされます。場合によっては、コストベースの最適化を実行する必要があることがあります。たとえば、クエリ処理時にマテリアライズドビューを考慮する場合、ビューのマッチングが発生します。ただし、ビューの一致が発生するのはコストベースの最適化中のみです。クエリに対してコストベースの最適化を実行しても、クエリ指定に単純なクエリのみを含める場合、FORCE OPTIMIZATION オプションを指定して、オプティマイザーが確実にクエリに対してコストベースの最適化を実行できるようにします。

同様に、プロシージャー内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャーに対するすべての呼び出しにオプティマイザーが強制的に使用されます。この場合、文のプランはキャッシュされません。

○ **FORCE NO OPTIMIZATION 句**　FORCE NO OPTIMIZATION 句は、文でオプティマイザーをバイパスする場合に指定します。文が複雑すぎて (データベースオプションの設定またはスキーマやクエリの特徴などにより) このような処理が実行できない場合、文は失敗し、データベースサーバーはエラーを返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ ***option-name = option-value*** オプション設定を指定します。指定する設定は、現在の文にのみ適用され、ODBC 実行可能アプリケーションによる設定など、パブリックオプションまたはテンポラリオプションの設定よりも優先されます。サポートされるオプションは次のとおりです。

- 「isolation_level オプション」『SQL Anywhere サーバー データベース管理』
- 「max_query_tasks オプション」『SQL Anywhere サーバー データベース管理』
- 「optimization_goal オプション」『SQL Anywhere サーバー データベース管理』
- 「optimization_level オプション」『SQL Anywhere サーバー データベース管理』
- 「optimization_workload オプション」『SQL Anywhere サーバー データベース管理』
- 「user_estimates オプション」『SQL Anywhere サーバー データベース管理』

クエリで isolation_level オプションを指定すると、そのクエリで指定した値が、現在のクエリの他のどの独立性レベル設定 (データベースの isolation_level オプションの設定やカーソルの設定など) よりも優先されます。

sequence-expression シーケンスジェネレーターから現在の値 (CURRVAL) または次の値 (NEXTVAL) を選択できます。

備考

SELECT 文は、次の目的で使用できます。

- データベースから結果を取得する。
- Interactive SQL で、データベース内のデータをブラウズ、またはデータベースから外部ファイルにデータをエクスポートする。
- プロシージャーとトリガー、または Embedded SQL 内で使用する。INTO 句のある SELECT 文を使ってデータベースから結果を取り出します。このとき、SELECT 文はローを 1 つだけ返します。複数のローを対象にクエリを実行する場合は、カーソルを使います。
- プロシージャーから結果セットを返す。

パーミッション

指定したテーブルとビューに対する SELECT パーミッションが必要です。

シーケンスジェネレーターから CURRVAL 値または NEXTVAL 値を選択するには、DBA 権限が必要です。または、シーケンスの所有者であるか、シーケンスジェネレーターを使用するためのパーミッションが付与されている必要があります。

関連する動作

なし

参照

- 「クエリ処理のフェーズをスキップするための条件」『SQL Anywhere サーバー SQL の使用法』
- 「高度：クエリ処理のフェーズ」『SQL Anywhere サーバー SQL の使用法』
- 「式」12 ページ
- 「FROM 句」758 ページ
- 「GROUP BY 句」788 ページ
- 「WINDOW 句」989 ページ
- 「探索条件」34 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」686 ページ
- 「UNION 文」964 ページ
- 「EXCEPT 文」738 ページ
- 「INTERSECT 文」810 ページ
- WITH table-hint 句、FROM 句 764 ページ
- 「materialized_view_optimization オプション」『SQL Anywhere サーバー データベース管理』
- 「ansi_update_constraints オプション」『SQL Anywhere サーバー データベース管理』
- 「reserved_keywords オプション」『SQL Anywhere サーバー データベース管理』
- 「カラム '%1' が見つかりません。」『エラーメッセージ』
- 「SELECT、UPDATE、DELETE クエリブロック内のロー制限句」『SQL Anywhere サーバー SQL の使用法』
- 「ジョイン：複数テーブルからのデータ検索」『SQL Anywhere サーバー SQL の使用法』
- 「共通テーブル式」『SQL Anywhere サーバー SQL の使用法』
- 「ユニークな値を生成するためのシーケンスの使用」『SQL Anywhere サーバー SQL の使用法』
- 「FOR XML EXPLICIT の使用」『SQL Anywhere サーバー SQL の使用法』
- 「クエリ結果を XML として取り出すための FOR XML 句の使用」『SQL Anywhere サーバー SQL の使用法』
- 「ODBC カーソル特性」『SQL Anywhere サーバー プログラミング』
- 「更新可能な文」『SQL Anywhere サーバー プログラミング』
- 「SQL Anywhere のカーソル」『SQL Anywhere サーバー プログラミング』
- 「Transact-SQL の外部ジョイン (*= or =*)」『SQL Anywhere サーバー SQL の使用法』
- 「Transact-SQL と互換性のあるクエリ」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　コア機能。SELECT 文は複雑なため、個々の句を標準に照らしてチェックしてください。たとえば、GROUP BY 句で指定できる ROLLUP キーワードは、オプションの SQL/2008 言語機能 T431 の一部です。SQL Anywhere でサポートされている、オプションの SQL/2008 言語機能の一部を次に示します。

  ○WINDOW 句と WINDOW 集合関数は、オプションの SQL/2008 言語機能 T611 および T612 を構成します。

  ○シーケンス式は、機能 T176 の一部です。

  ○共通テーブル式は、オプションの SQL/2008 言語機能 T121 です。ネストされたクエリ式に含まれる共通テーブル式は、機能 T122 です。WITH RECURSIVE は、オプションの SQL/2008 言語機能 T131 です。ネストされたクエリに含まれている場合は、機能 T132 を構成します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○UNION、EXCEPT、または INTERSECT を含むクエリ式とともに ORDER BY 句を指定する機能は、オプションの機能 F850 です。サブクエリで ORDER BY を指定する機能は、機能 F851 です。

○SQL 標準では、FOR UPDATE と FOR READ ONLY はカーソル宣言の一部です。

SQL Anywhere では、SELECT 文の SQL/2008 定義に対する多くの拡張がサポートされています。その一部を次に示します。

○オプションの *cursor-concurrency* 句 (FOR UPDATE BY { LOCK | VALUES | TIMESTAMP})はベンダー拡張です。

○FOR XML 句、OPTION 句、INTO 句はベンダー拡張です。

○row-limitation 句はベンダー拡張です。SQL/2008 標準では、row-limitation は、オプションの言語機能 F856 である FETCH FIRST 構文を使用してサポートされます。機能 F856 の構文は SQL Anywhere ではサポートされていません。

○ORDER BY *n* を指定する機能はベンダー拡張です。

○SQL/2008 では、INSENSITIVE カーソル以外のすべてのカーソルがデフォルトで更新可能になっています。Embedded SQL プログラムでデフォルトで使用される読み込み専用カーソルはベンダー拡張です。

● **Transact-SQL**　SQL Anywhere と Adaptive Server Enterprise での SELECT 文のサポートには大きな違いがあります。SELECT 文のいくつかの機能は、Adaptive Server Enterprise ではサポートされていません。

これらの違いを次に示します。

○Sybase ASE では、SQL Anywhere の cursor-concurrency 句はサポートされていません。フェッチされたローのロックを取得するには、HOLDLOCK テーブルヒントを使用します。

○Adaptive Server Enterprise では、再帰クエリも共通テーブル式もサポートされていません。

○Adaptive Server Enterprise と SQL Anywhere では、Transact-SQL 外部ジョインに関して違いがあります。

Transact-SQL では、Watcom SQL SET 文ではなく、SELECT 文を使用して値を変数に割り当てます。

例

次のクエリは、Employees テーブルの全従業員数を返します。

```
SELECT COUNT(*)
FROM Employees;
```

次のクエリは、すべての顧客とその顧客からの注文の総額をリストします。

```
SELECT CompanyName,
  CAST( SUM( SalesOrderItems.Quantity *
  Products.UnitPrice ) AS INTEGER ) VALUE
```

```
FROM Customers
  JOIN SalesOrders
  JOIN SalesOrderItems
  JOIN Products
GROUP BY CompanyName
ORDER BY VALUE DESC;
```

次の文は、Embedded SQL SELECT 文を示します。Employees テーブル内の従業員数が選択され、:size ホスト変数に格納されます。

```
SELECT COUNT(*) INTO :size
FROM Employees;
```

次の文は、結果セットの最初のローを迅速に返すように最適化されています。

```
SELECT Name
FROM Products
GROUP BY Name
HAVING COUNT( * ) > 1
AND MAX( UnitPrice ) > 10
OPTION( optimization_goal = 'first-row' );
```

SET CONNECTION statement [Interactive SQL] [ESQL]

アクティブなデータベース接続を変更します。

構文

SET CONNECTION [*connection-name*]

connection-name :
identifier
| *string*
| *hostvar*

備考

SET CONNECTION 文は、アクティブなデータベース接続を connection-name に変更します。現在の接続状態を保存し、再びアクティブな接続になるときにこれを再開します。connection-name が省略され、指定されない接続がある場合は、この接続がアクティブな接続になります。

カーソルを Embedded SQL で開くとき、カーソルを現在の接続と関連付けます。接続が変更されると、前のアクティブな接続のカーソル名にはアクセスできません。これらのカーソルはアクティブなまま配置され、関連付けられている接続が再びアクティブになると、アクセスできるようになります。

パーミッション

なし

関連する動作

なし

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CONNECT 文 [ESQL] [Interactive SQL]」516 ページ
- 「DISCONNECT 文 [ESQL] [Interactive SQL]」707 ページ
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　SET CONNECTION は、オプションの SQL/2008 言語機能 F771、"Connection management" の一部です。Interactive SQL セッション内での使用はベンダー拡張です。

例

次の例は、Embedded SQL での記述です。

```
EXEC SQL SET CONNECTION :conn_name;
```

Interactive SQL から、現在の接続を接続名 conn1 に設定します。

```
SET CONNECTION conn1;
```

SET DESCRIPTOR 文 [ESQL]

SQL 記述子領域内の変数を記述したり、データを記述子領域に格納したりします。

構文

SET DESCRIPTOR *descriptor-name*
{ **COUNT** = { *integer* | *hostvar* }
| **VALUE** { *integer* | *hostvar* } *assignment*, ... }

assignment :
{ **TYPE** | **SCALE** | **PRECISION** | **LENGTH** | **INDICATOR** }
　= { *integer* | *hostvar* }
| **DATA** = *hostvar*

descriptor-name : *identifier*

備考

SET DESCRIPTOR 文は、記述子領域内の変数を記述したり、データを記述子領域に格納するために使用します。

SET...COUNT 文を使用すると、記述子領域内の記述される変数の数を設定できます。カウントの対象となる変数の数は、記述子領域を割り付けたときに指定した変数の数を超えることはできません。

値 { *integer* | *hostvar* } は、代入の対象となる記述子領域内の変数を指定します。

SET...DATA の実行中に型チェックが実行され、記述子領域の変数とホスト変数のデータ型が同じかどうか確認されます。LONG VARCHAR と LONG BINARY は SET DESCRIPTOR...DATA ではサポートされていません。

エラーが発生すると、エラーコードが SQLCA に返されます。

パーミッション

なし

関連する動作

なし

参照

- 「ALLOCATE DESCRIPTOR 文 [ESQL]」406 ページ
- 「DEALLOCATE DESCRIPTOR 文 [ESQL]」684 ページ
- 「SQLDA (SQL descriptor area)」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　SET DESCRIPTOR は、オプションの SQL/2008 言語機能 B031、"Basic dynamic SQL" の一部です。

例

次の例は、sqlda の位置 col_num にあるカラムの型を設定します。

```
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;
}
```

SET MIRROR OPTION 文

> **注意**
> 読み込み専用のスケールアウトとデータベースミラーリングには、それぞれ別のライセンスが必要です。 「別途ライセンスが必要なコンポーネント」『SQL Anywhere 12 紹介』を参照してください。

データベースミラーリングと読み込み専用のスケールアウトの設定を制御するオプションの値を変更します。

構文

SET MIRROR OPTION *option-name*={ *option-value* | **NULL** }

option-name :
authentication_string
auto_add_fan_out
auto_add_server
auto_failover
child_creation
page_timeout
max_disconnected_time

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

max_retry_connect_time
synchronization_mode

パラメーター

option-name	適用対象	値	デフォルト	説明
authentication_string	データベースミラーリング	文字列	null	データベースミラーリングシステムのすべてのサーバーで使用される認証文字列を指定します。データベースミラーリングには、認証文字列は必須です。
auto_add_fan_out	読み込み専用のスケールアウト	整数	10	分岐ごとの子の最大数を指定します。指定できる最小の値は 2 です。
auto_add_server	読み込み専用のスケールアウト	文字列	null	自動割り当てツリーの親として動作するデータベースサーバーの名前を指定します。

option-name	適用対象	値	デフォルト	説明
auto_failover	データベースミラーリング	on、off	null	現在のプライマリサーバーで障害が発生した場合にミラーサーバーが自動的にプライマリサーバーになるかどうかを指定します。このオプションは同期モードには適用されません。 このオプションでは、ブール値を使用できます (YES、ON、TRUE、または 1 の場合は自動フェールオーバーがオンになり、NO、OFF、FALSE、または 0 の場合は自動フェールオーバーがオフになります)。パラメーターは大文字と小文字が区別されません。 非同期モードまたは非同期フルページモードを使用している場合は、auto_failover オプションを on に設定することをおすすめします。それによって、プライマリサーバーで障害が発生した場合、ミラーサーバーが自動的に

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

option-name	適用対象	値	デフォルト	説明
				プライマリサーバーとなります。
child_creation	読み込み専用のスケールアウト	automatic、off、manual	automatic	コピーノードを自動的に作成するかどうかを制御します。
page_timeout	データベースミラーリング	整数 (秒)	5	トランザクションログページを満杯かどうかにかかわらずミラーサーバーに送信する間隔を秒単位で指定します。このオプションは、非同期フルページモードを使用している場合にのみ適用されます。
max_disconnected_time	読み込み専用のスケールアウト	整数 (秒)	時間制限なし	親接続が失われた後、コピーサーバーがルートデータベースサーバーに接続を試みる期間を指定します。指定した期間内にコピーサーバーが接続できなかった場合、データベースは停止します。
max_retry_connect_time	読み込み専用のスケールアウト	整数 (秒)	120	親が使用できなくなった後、コピーノードが親に再接続を試みる期間を指定します。

option-name	適用対象	値	デフォルト	説明
synchronization_mode	データベースミラーリング	synchronous、asynchronous、asyncfullpage	synchronous	データベースミラーリングで使用する同期実行モードとして、同期 (sync)、非同期 (async)、非同期フルページ (page) のいずれかを指定します。同期モードでは、ミラーサーバーでトランザクションを記録するタイミングと方法を制御します。

備考

CREATE MIRROR SERVER 文を使用して、データベースミラーリングシステムまたは読み込み専用のスケールアウトシステム用のデータベースサーバーを作成した後、SET MIRROR OPTION 文を使用してそのシステムの設定を指定できます。

パーミッション

DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「コピーノードへの親の自動割り当て」『SQL Anywhere サーバー データベース管理』
- 「親接続の消失の処理」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリングモード」『SQL Anywhere サーバー データベース管理』
- 「子コピーノードを追加する方法」『SQL Anywhere サーバー データベース管理』
- 「-xa dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリング」『SQL Anywhere サーバー データベース管理』
- 「SQL Anywhere の読み込み専用のスケールアウト」『SQL Anywhere サーバー データベース管理』
- 「CREATE MIRROR SERVER 文」579 ページ
- 「ALTER MIRROR SERVER 文」430 ページ
- 「DROP MIRROR SERVER 文」719 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、データベースミラーリングシステムの認証文字列を abc に設定します。

```
SET MIRROR OPTION authentication_string = 'abc';
```

SET OPTION 文

データベースオプションと接続オプションの値を変更します。

構文 1

SET [**EXISTING**] [**TEMPORARY**] **OPTION**
[*userid*.| **PUBLIC**.]*option-name* = [*option-value*]

構文 2 (旧式)

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*

パラメーター

option-value 構文 1 では、*option-value* は次のいずれかです。

- キーワード ON、OFF、または NULL
- string literal 値 (一重引用符で囲む)
- 任意の有効なフォーマット (NUMERIC など) の number
- Embedded SQL プログラム内では、ホスト変数 hostvar の値
- **@** 記号で始まる変数名を持つ SQL 変数の値

構文 2 では、任意の有効な識別子をオプション値として指定できます。また、識別子の名前は、一重引用符で囲んだ文字列リテラルであるかのように扱われます。たとえば、次の 2 つの文は同等です。

```
SET TEMPORARY OPTION ansi_update_constraints = 'strict';
```

```
SET TEMPORARY OPTION ansi_update_constraints = strict;
```

備考

SET OPTION 文を使用すると、データベースサーバーの動作に影響を及ぼすオプションを変更できます。オプションの値を設定すると、すべてのユーザー (パブリック)、個々のユーザー、または現在の接続の動作を変更できます。新しい設定は、テンポラリ設定にも永久設定にもできます。

SET OPTION 文で設定できるオプションのクラスは次のとおりです。

- Transact-SQL 互換性オプション。
- 接続オプションとデータベースオプション。
- 同期オプション。
- ユーザー定義のオプション

オプションのスコープ　ほとんどのオプションでは、3 つのレベルのスコープ (パブリック、ユーザー、接続) で値を設定できます。login_mode などの一部の特定のオプションはパブリックレベルのみに制限されます。接続オプションは他の 2 つのレベルよりも優先され、ユーザーオプションはパブリックオプションよりも優先されます。接続レベルのオプションは、TEMPORARY キーワードを使用して設定します。現在のユーザーに対してユーザーレベルのオプションを設定すると、対応する接続レベルのオプションも同時に設定されます。

ユーザー ID を指定すると、そのユーザーにオプション値が適用されます。PUBLIC に指定すると、オプション値はそのオプションに個々の設定を持たないすべてのユーザーに適用されます。デフォルトでは、オプション値は SET OPTION 文を実行した現在ログオン中のユーザー ID に適用されます。

たとえば、次の文は、DBA が SQL 文を実行しているユーザーである場合、ユーザー DBA にオプションの変更を適用します。

```
SET OPTION precision = 40;
```

ただし、次の文は、すべてのユーザーが属するグループである PUBLIC ユーザー ID に変更を適用します。

```
SET OPTION PUBLIC.login_mode = 'Standard';
```

TEMPORARY オプション　デフォルトでは、TEMPORARY キーワードを指定しないかぎり、新しいオプション値は永久値になります。SET OPTION 文に TEMPORARY キーワードを追加すると、変更の適用期間に影響します。

SET TEMPORARY OPTION 文がユーザー ID で修飾されていない場合、新しいオプション値は現在の接続のみで有効です。

PUBLIC ユーザー ID を使用して SET TEMPORARY OPTION を使用すると、データベースの実行中はその変更が継続します。データベースを停止すると、PUBLIC グループの TEMPORARY オプションは永久値に戻ります。

PUBLIC ユーザー ID のテンポラリオプションを設定すると、セキュリティの利点を提供します。たとえば、login_mode オプションが有効なときは、データベースは実行中のシステムのログインセキュリティに依存します。このオプションを一時的に有効にすると、Windows ドメインのセキュリティに依存しているデータベースは、データベースが停止し、ローカルコンピューターにコピーされるといった場合でも、危険にさらされることはありません。この場合、login_mode オプションを一時的に有効にしてあると、オプションは永久値の Standard に戻ります。このモードでは、統合化ログインを使用できません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプション設定の削除　*option-value* を省略すると、指定したオプション設定がデータベースから削除されます。これがユーザーレベルのオプション設定の場合、値は PUBLIC 設定に戻ります。TEMPORARY オプションを削除すると、オプション設定はそのユーザーの永久設定に戻ります。

オプションのデータ型　オプションは、ブール値、数値、または文字列値にできますが、データベースには常に文字列として格納されます。オプション設定は、プロパティ関数の結果として返される場合や、関数またはシステムストアドプロシージャーの結果として返される場合、常に文字列として返されます。オプション値は、データベースページサイズよりも大きい値には設定できません。

ユーザー定義のオプション　どのようなオプションでも、ユーザー定義であるかどうかにかかわらず、パブリック設定がなければユーザー固有の値を割り当てることはできません。データベースサーバーでは、ユーザー定義オプションへの TEMPORARY 値の設定はサポートされていません。たとえば、ApplicationControl という名前のユーザー定義のオプションを作成するには、まず次の文を実行します。

```
SET OPTION PUBLIC.ApplicationControl = 'Default';
```

この文は、すべてのユーザーに対して ApplicationControl オプションを Default に設定し、サーバーへの新しい接続ごとに有効にします。その後、個々のユーザーで別個に SET OPTION 文を実行して、このオプションの独自の設定を確立できます。

制限　PUBLIC ユーザー ID または他のデータベースユーザーに対してオプションを設定できるのは、DBA 権限を持つユーザーのみです。

EXISTING キーワードを使用した場合、個別のユーザー ID のオプション値は、そのオプションにすでに PUBLIC ユーザー ID 設定が行われていないかぎり、設定できません。

警告
オープンカーソルからローをフェッチしているときにオプション設定を変更しないでください。変更すると、動作が不適切に定義され、セマンティックが保証されなくなる可能性があります。たとえば、カーソルからフェッチしている間に date_format 設定を変えると、結果セットのロー同士の日付フォーマットが異なってしまいます。

接続またはユーザーの特定のオプションの値に対するクエリを実行するには、いくつかの方法があります。

SET OPTION 文は、SQL Flagger で無視されます。

パーミッション

ユーザー自身のオプションを設定するには、権限は必要ありません。

別のユーザーまたは PUBLIC に対してデータベースオプションを設定するには、DBA 権限が必要です。

関連する動作

TEMPORARY を指定しない場合、オートコミットが実行されます。

参照

- 「データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「互換性オプション」『SQL Anywhere サーバー データベース管理』
- 「同期オプション」『SQL Anywhere サーバー データベース管理』
- 「SQL Remote オプション」『SQL Anywhere サーバー データベース管理』
- 「オプション設定の検索」『SQL Anywhere サーバー データベース管理』
- 「データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「SYSOPTION システムビュー」1285 ページ
- 「sa_conn_options システムプロシージャー」1060 ページ
- 「sa_conn_options システムプロシージャー」1060 ページ
- 「CONNECTION_PROPERTY 関数 [システム]」178 ページ
- 「GET OPTION 文 [ESQL]」774 ページ
- 「SET OPTION 文 [Interactive SQL]」916 ページ
- 「SET 文 [T-SQL]」929 ページ
- 「SET REMOTE OPTION 文 [SQL Remote]」919 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

個別に設定することなく、すべてのユーザーの日付フォーマットオプションを設定します。

```
SET OPTION PUBLIC.date_format = 'Mmm dd yyyy';
```

wait_for_commit オプションを On に設定します。

```
SET OPTION wait_for_commit = 'On';
```

次は、Embedded SQL の例です。

```
EXEC SQL SET TEMPORARY OPTION date_format = :value;
```

現在接続されているユーザーに対して date_format オプションを設定します。同じユーザー ID による以降の接続では、このオプション値が使用されます。

```
SET OPTION date_format = 'yyyy/mm/dd';
```

次の文は、現在のユーザー ID の date_format オプションを削除します。この文の実行後、date_format の設定としては PUBLIC が代わりに使用されます。

```
SET OPTION date_format=;
```

SET OPTION 文 [Interactive SQL]

Interactive SQL オプションの値を変更します。

構文 1 - Interactive SQL オプションの設定

SET OPTION *option-name* = [*option-value*] | **SET TEMPORARY OPTION** *option-name* = [*option-value*]

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

option-name : *identifier* | *string* | *hostvar*

option-value : *string* | *identifier* | *number*

構文 2 - 現在の Interactive SQL オプションの永久的な保存

SET PERMANENT

構文 3 - 現在のデータベースオプション設定の表示

SET

備考

SET OPTION 構文を使用してオプションを設定すると、そのオプション設定は永久的に格納され、他の SET OPTION 文によって変更されるまで変わりません。

SET TEMPORARY OPTION 構文を使用すると、オプション設定を一時的に変更できます。一時的な設定は、Interactive SQL を閉じるまで有効です。Interactive SQL を次回起動するとき、オプションはその恒久的な設定に戻ります。

SET PERMANENT 構文を使用すると、現在の Interactive SQL オプション設定がすべて永久的に保存されます (一時的な設定もすべて永久的になります)。

option-value を省略すると、指定したオプションはデフォルト値に設定されます。

構文 3 を使用すると、現在のすべてのデータベースオプション設定がウィンドウに表示されます。データベースサーバーにテンポラリオプション設定がある場合、永久設定ではなく、テンポラリ設定が表示されます。

Interactive SQL のオプション設定は、データベースにではなく、クライアントコンピューターに保存されます。

次の表は、Interactive SQL のオプションを示します。

オプション	値	デフォルト
「auto_commit オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off	Off
「auto_refetch オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off	On
「bell オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off	On
「command_delimiter オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	文字列	';'

オプション	値	デフォルト
「commit_on_exit オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off	On
「default_isql_encoding オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	文字列	空の文字列
「echo オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off	On
「input_format オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	TEXT、FIXED	TEXT
「isql_allow_read_client_file オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off、Prompt	Prompt
「isql_allow_write_client_file オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off、Prompt	Prompt
「isql_command_timing オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off	On
「isql_escape_character オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	文字	'¥'
「isql_field_separator オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	文字列	','
「isql_maximum_displayed_rows オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	All または負でない整数	500
「isql_print_result_set オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	Last、All、None	Last
「isql_quote オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	文字列	'

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプション	値	デフォルト
「isql_show_multiple_result_sets オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	On、Off	Off
「nulls オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	文字列	(NULL)'
「on_error オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	Stop、Continue、Prompt、Exit、Notify_Continue、Notify_Stop、Notify_Exit	Prompt
「output_format オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	TEXT、FIXED、HTML、SQL、XML	TEXT
「output_length オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	整数	0
「output_nulls オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	文字列	空の文字列
「truncation_length オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』	整数	256

標準と互換性

- **SQL/2008**　ベンダー拡張。

参照

- 「Interactive SQL オプション」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

例

次の文は、on_error オプションの値を変更します。

```
SET OPTION on_error='continue';
```

SET REMOTE OPTION 文 [SQL Remote]

SQL Remote メッセージリンクのメッセージ制御パラメーターを設定します。

構文

SET REMOTE *link-name* **OPTION**
[*userid*.| **PUBLIC**.]*link-option-name* = *link-option-value*

link-name :
file
| **ftp**
| **http**
| **smtp**

link-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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| **pop3_host**
| **pop3_password**
| **pop3_userid**
| **smtp_authenticate**
| **smtp_option**
| **smtp_password**
| **smtp_userid**
| **suppress_dialogs**
| **top_supported**

link-option-value : *string*

パラメーター

userid　*userid* を指定しない場合、現在のパブリッシャーが使用されます。

common-options　FILE、FTP、HTTP、SMTP メッセージシステムに共通のオプションは、次のとおりです。

○ **debug**　このパラメーターは、YES または NO に設定されます。デフォルトは NO です。YES を設定すると、メッセージシステムに固有のデバッグ出力が表示されます。この情報は、メッセージシステムのトラブルシューティングに使用できます。

○ **max_retries**　デフォルトでは、SQL Remote が継続モードで実行されていて、メッセージシステムにアクセスするときにエラーが発生する場合、送受信フェーズの後に停止します。このパラメーターを使用して、停止前に SQL Remote が再試行する送受信フェーズの回数を指定します。

○ **output_log_send_on_error**　エラーが発生すると、情報を送信します。

○ **output_log_send_limit**　統合データベースに送信される情報の量を制限します。output_log_send_limit オプションには、統合データベースに送信されるバイト数を指定します。これは、出力ログの最後に表示されます (つまり、最後のエントリです)。デフォルトは 5 K です。

○ **output_log_send_now**　YES に設定されると、統合データベースに出力ログ情報を送信します。次のポーリング時にリモートデータベースは出力ログ情報を送信し、その後、output_log_send_now オプションが NO にリセットされます。

○ **pause_after_failure**　このパラメーターが使えるのは、max_retries がゼロ以外の値に指定され、SQL Remote が継続モードで実行されている場合です。メッセージシステムでエラーが発生すると、このパラメーターは、SQL Remote が送受信フェーズを再試行する間に待機する時間を秒で指定します。

○ **encode_dll**　カスタムエンコードスキームを実装している場合は、作成したカスタムエンコード DLL のフルパスをこの値に設定する必要があります。

file-options　これらのオプションは、ファイルメッセージシステムにのみ適用されます。

○ **directory**　メッセージが格納されるディレクトリです。このパラメーターは、SQLREMOTE 環境変数の代替となります。

○ **invalid_extensions** メッセージングシステムでのファイルの生成時に SQL Remote Message Agent (dbremote) で使用されないようにするファイル拡張子の、カンマで区切られたリスト。

○ **Unlink_delay** ファイル削除に失敗した後、次にファイルの削除を試みるまでの秒数。unlink_delay に対して値が定義されていない場合、デフォルト動作は、最初の試行失敗の後に 1 秒間、2 回目の試行失敗の後に 2 秒間、3 回目の試行失敗の後に 3 秒間、4 回目の試行失敗の後に 4 秒間一時停止するように設定されています。

ftp-options これらのオプションは、FTP メッセージシステムにのみ適用されます。

○ **active_mode** このパラメーターは、SQL Remote がクライアント／サーバー接続を確立する方法を制御します。このパラメーターは、YES または NO に設定されます。デフォルトは NO (受動モード) です。受動モードは推奨の転送モードで、FTP メッセージリンクにデフォルトとして設定されています。受動モードでは、すべてのデータ転送接続はクライアント (この場合はメッセージリンク) から開始されます。アクティブモードでは、FTP サーバーがすべてのデータ接続を開始します。

○ **host** FTP サーバーを実行しているコンピューターのホスト名。このパラメーターには、ホスト名 (FTP.ianywhere.com など) または IP アドレス (192.138.151.66 など) を使用できます。

○ **invalid_extensions** メッセージングシステムでのファイルの生成時に dbremote で使用されないようにするファイル拡張子の、カンマで区切られたリスト。

○ **password** FTP ホストにアクセスするためのパスワード。

○ **port** FTP の接続に使用される IP ポート番号。このパラーメータは通常は不要です。

○ **reconnect_retries** 失敗になる前に、リンクがサーバーでソケットを開こうと試行する回数。デフォルト値は 4 です。このパラメーターを設定すると、再接続のみに影響します。FTP リンクによる最初の接続には影響しません。

○ **reconnect_pause** 接続の試行失敗後に次の接続まで待機する秒数。デフォルトは 30 秒です。このパラメーターを設定すると、再接続のみに影響します。FTP リンクによる最初の接続には影響しません。

○ **root_directory** メッセージが保存される、FTP ホストサイトのルートディレクトリ。

○ **suppress_dialogs** このパラメーターは、TRUE または FALSE に設定されます。TRUE に設定されている場合は、FTP サーバーへの接続の試行失敗後に、**[接続]** ウィンドウは表示されません。代わりに、エラーが発生します。

○ **user** FTP ホストにアクセスするためのユーザー名。

http-options これらのオプションは、HTTP メッセージシステムにのみ適用されます。

○ **certificate** 安全な (HTTPS) 要求を行うには、HTTPS サーバーで使用される証明書にクライアントがアクセスできる必要があります。必要な情報は、セミコロンで区切られたキー／値のペアの文字列で指定されます。ファイルキーを使用して証明書のファイル名を指定できます。ファイルキーと証明書キーを一緒に指定することはできません。次のキーを使用できます。

Key	Abbreviation	Description
file		証明書のファイル名
certificate	cert	証明書自体
company	co	証明書で指定された会社
unit		証明書で指定された会社の部署
name		証明書で指定された通称

証明書は、HTTPS サーバーに対する要求、または安全でないサーバーから安全なサーバーにリダイレクトされる可能性がある要求に対してのみ必要です。PEM でフォーマットされた証明書のみがサポートされています。**certificate='***file=filename***'**

○ **client_port**　SQL Remote が HTTP を使用して通信するポート番号を識別します。これは「送信」TCP/IP 接続をフィルターするファイアウォール経由で接続するためのもので、この用途にかぎり推奨されています。単一のポート番号、ポート番号の範囲、または両方の組み合わせを指定できます。指定したクライアントポートの数が少ないと、SQL Remote が前回の実行でポートを閉じた後すぐにオペレーティングシステムがポートを解放しなかった場合に、SQL Remote でメッセージの送受信ができなくなる可能性があります。

○ **debug**　YES に設定すると、すべての HTTP コマンドと応答が出力ログに表示されます。この情報は、HTTP のサポート問題のトラブルシューティングに使用できます。デフォルトは NO です。

○ **https**　HTTPS (**https=yes**) または HTTP (**https=no**) を使用するかどうかを指定します。

○ **password**　メッセージサーバーのデータベースパスワード。パスワードは、RFC 2617 の基本認証を使用してサードパーティの HTTP サーバーとゲートウェイに対する認証を行います。

○ **proxy_host**　プロキシサーバーの URI を指定します。SQL Remote がプロキシサーバーを介してネットワークにアクセスする場合に使用します。SQL Remote がプロキシサーバーに接続し、そのプロキシサーバーを介してメッセージサーバーに要求を送信することを示します。

'

○ **reconnect_retries**　失敗になる前に、リンクがサーバーでソケットを開こうと試行する回数。デフォルト値は 4 です。このパラメーターを設定すると、再接続のみに影響します。FTP リンクによる最初の接続には影響しません。

○ **reconnect_pause**　接続の試行失敗後に次の接続まで待機する秒数。デフォルトは 30 秒です。このパラメーターを設定すると、再接続のみに影響します。FTP リンクによる最初の接続には影響しません。

○ **root_directory**　この HTTP 制御パラメーターは、クライアント側で指定された場合には無視されます。sr_add_message_server または sr_update_message_server ストアドプロシージャーを呼び出す前に、メッセージサーバーでこの制御パラメーターを定義します。HTTP メッセージシステムを使用する場合、リモートユーザーまたはパブリッシャーに指定するアドレスには、1 つのサブディレクトリのみを含めることができます。複数のサブディレクトリを含めることはできません。

○ **url**　サーバー名または IP アドレスを指定し、任意で、使用する HTTP サーバーのポート番号をセミコロンで区切って指定します。要求が Relay Server を経由する場合は、URL 拡張子を任意で追加して、要求の渡し先のサーバーファームを示すことができます。

○ **user**　メッセージサーバーのデータベースユーザー ID。RFC 2617 の基本認証を使用してサードパーティの HTTP サーバーとゲートウェイに対する認証を行います。

smtp-options　これらのオプションは、SMTP メッセージシステムにのみ適用されます。

○ **local_host**　ローカルコンピューターの名前。SQL Remote がローカルホスト名を決定できないコンピューターで設定すると便利です。ローカルホスト名は、任意の SMTP サーバーとのセッションを開始するのに必要です。ほとんどのネットワーク環境では、ローカルホスト名が自動的に決定されるため、このエントリは不要です。

○ **pop3_host**　POP ホストを実行しているコンピューターの名前。一般的には、SMTP ホストと同じ名前です。SMTP/POP3 ログインウィンドウの POP3 ホストフィールドに対応しています。

○ **pop3_password**　メールの受信に使用するパスワード。SMTP/POP3 ログインウィンドウのパスワードフィールドに対応しています。

○ **pop3_userid**　メールの受信に使用するユーザー ID。POP ユーザー ID は、SMTP/POP3 ログインウィンドウのユーザー ID フィールドに対応しています。必ず POP ホスト管理者からユーザー ID を取得してください。

○ **smtp_host**　SMTP サーバーが動作しているコンピューターの名前。SMTP/POP3 ログインウィンドウの SMTP ホストフィールドに対応しています。

○ **top_supported**　受信メッセージを列挙するときに、SQL Remote は TOP という POP3 コマンドを使用します。TOP コマンドは、すべての POP サーバーでサポートされているわけではありません。top_supported パラメーターを NO に設定すると、SQL Remote は RETR コマンドを使用します。このコマンドは TOP よりも効率は落ちますが、すべての POP サーバーで動作します。デフォルトは YES です。

○ **smtp_authenticate**　SMTP リンクがユーザーを認証するかどうかを決定します。デフォルト値は YES です。SMTP 認証を無効にする場合は、このパラメーターを NO に設定します。

○ **smtp_userid**　SMTP 認証のユーザー ID。デフォルトでは、このパラメーターは pop3_userid パラメーターと同じ値を取ります。smtp_userid は、ユーザー ID が POP サーバー上のユーザー ID と異なる場合にのみ設定する必要があります。

○ **smtp_password**　SMTP 認証のパスワード。デフォルトでは、このパラメーターは pop3_password パラメーターと同じ値を取ります。smtp_password は、ユーザー ID が POP サーバー上のユーザー ID と異なる場合にのみ設定する必要があります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **suppress_dialogs**　このパラメーターが true に設定されている場合は、メールサーバーへの接続の試行失敗後に、**[接続]** ウィンドウは表示されません。代わりに、エラーが発生します。

備考

メッセージリンクの初回使用時に、ユーザーがメッセージリンクパラメーターを [メッセージリンク] ウィンドウに入力すると、SQL Remote (dbremote) Message Agent はそのパラメーターを保存します。その場合はこの文を明示的に使用する必要はありません。この文は、たくさんのデータベースから抽出を行うための統合データベースを作成する場合に非常に効果的です。

オプション名では、大文字と小文字が区別されます。オプションの値で大文字と小文字が区別されるかどうかは、オプションによって異なります。ブール値の場合、大文字と小文字は区別されません。パスワード、ディレクトリ名、その他の文字列では、ファイルシステム (ディレクトリ名の場合) またはデータベース (ユーザー ID とパスワードの場合) で大文字と小文字を区別するかどうかに従います。

パーミッション

DBA 権限。パブリッシャーは自分自身のオプションを設定できます。

関連する動作

オートコミット。

参照

- 「リモートデータベースからのエラーの収集」『SQL Remote』
- 「リモートメッセージタイプ制御パラメーターの設定」『SQL Remote』
- 「カスタムエンコードスキーム」『SQL Remote』
- 「SET OPTION 文」913 ページ
- 「FTP メッセージシステム」『SQL Remote』
- 「FILE メッセージシステム」『SQL Remote』
- 「HTTP メッセージシステム」『SQL Remote』
- 「チュートリアル：HTTP メッセージシステムを使用したレプリケーションシステムの設定」『SQL Remote』
- 「SMTP メッセージシステム」『SQL Remote』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、ユーザー myuser 用の FTP リンクに対し、FTP ホストを *ftp.mycompany.com* に設定します。

```
SET REMOTE FTP OPTION myuser.host = 'ftp.mycompany.com';
```

次の文は、生成されるメッセージの特定ファイル拡張子を SQL Remote が使用しないようにします。

```
SET REMOTE FTP OPTION "Public"."invalid_extensions"='exe,pif,dll,bat,cmd,vbs';
```

次の文は、ユーザー myuser 用の HTTP リンクのローカルホストを指すように URL を設定します。

```
SET REMOTE HTTP OPTION myuser.url='localhost:8033';
```

次の文は、要求を srhttp ファームに転送する Relay Server を指すように HTTP URL を設定します。

```
SET REMOTE HTTP OPTION "public"."url"='iis7.company.com:80/rs/client/rs_client.dll/srhttp';
```

SET SQLCA 文 [ESQL]

デフォルトのグローバル *sqlca* 以外の SQLCA を使用するように、SQL プリプロセッサーに通知します。

構文

SET SQLCA *sqlca*

sqlca : *identifier* | *string*

備考

SET SQLCA 文は、デフォルトのグローバル *sqlca* 以外の SQLCA を使うように、SQL プリプロセッサーに通知します。sqlca は、SQLCA ポインターに対する C 言語リファレンスである識別子または文字列を指定します。

Embedded SQL 文ごとに、現在の SQLCA ポインターを暗黙のうちにデータベースインターフェイスライブラリに渡します。C 言語ソースにおいて、この文の後に記述されているすべての Embedded SQL 文は、新しい SQLCA を使います。

この文を使用するのは、再入可能コードを記述するときのみです。

sqlca は、ローカル変数を参照しなければなりません。グローバルまたはモジュール静的変数は別のスレッドによる修正を受けます。

パーミッション

なし

関連する動作

なし

参照

- 「マルチスレッドまたは再入可能コードでの SQLCA 管理」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008** ベンダー拡張。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

Windows DLL には独自の関数を含めることもできます。DLL を使用する各アプリケーションは独自の SQLCA を持ちます。

```
an_sql_code FAR PASCAL ExecuteSQL( an_application *app, char *com )
{
  EXEC SQL BEGIN DECLARE SECTION;
  char *sqlcommand;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL SET SQLCA "&app->.sqlca";
  sqlcommand = com;
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL EXECUTE IMMEDIATE :sqlcommand;
return( SQLCODE );
}
```

SET 文

SQL 変数に値を代入します。

構文

SET *identifier* **=** *expression*

備考

SET 文は新しい値を変数に代入します。この変数は、CREATE VARIABLE 文または DECLARE 文を使って事前に作成されたものであるか、プロシージャーの OUTPUT パラメーターであることが必要です。この変数名は、オプションとして、名前の前に @ を付ける Transact-SQL の規則を使用できます。次に例を示します。

```
SET @localvar = 42
```

変数は、カラム名を使用できる場所なら SQL 文のどこででも使うことができます。変数と同じ名前のカラム名がある場合は、変数値を使用できます。

変数は、現在の接続に固有でローカルなもので、データベースとの接続を切断したり、DROP VARIABLE 文を使用すると自動的に消えます。変数は、COMMIT 文や ROLLBACK 文の対象になりません。

変数は、Embedded SQL プログラムから INSERT または UPDATE 文の対象となるサイズの大きなテキストまたはバイナリオブジェクトを作成するために必要です。これは、Embedded SQL のホスト変数のサイズが 32767 バイトに制限されているためです。

パーミッション

なし

関連する動作

なし

参照

- 「CREATE VARIABLE 文」679 ページ
- 「DECLARE 文」693 ページ
- 「SET 文 [T-SQL]」929 ページ
- 「DROP VARIABLE 文」736 ページ
- 「式」12 ページ
- 「ホスト変数の使用法」『SQL Anywhere サーバー プログラミング』

標準と互換性

- **SQL/2008**　SET 文は、オプションの SQL/2008 言語機能 P002、"Computational completeness" の一部です。

- **Transact-SQL**　SET 文は Adaptive Server Enterprise でサポートされています。ASE では、1 つの SET 文で個々の割り当て句をカンマで区切って、複数の変数に値を割り当てることができます。

例

この単純な例では、birthday という変数の作成を示し、SET を使用して日付を CURRENT DATE に設定します。

```
CREATE VARIABLE @birthday DATE;
SET @birthday = CURRENT DATE;
```

次のコードフラグメントは、データベースに大きいテキスト値を挿入します。

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_VARCHAR( 5000 ) buffer;
/* Note: maximum DECL_VARCHAR size is 32765 */
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE VARIABLE hold_blob LONG VARCHAR;
EXEC SQL SET hold_blob = '';
for(;;) {
  /* read some data into buffer ... */
  size = fread( buffer, 1, 5000, fp );
  if( size <= 0 ) break;
  /* Does not work if data contains null chars */
  EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES( 1, hold_blob );
EXEC SQL DROP VARIABLE hold_blob;
```

次のコードフラグメントは、データベースに大きいバイナリ値を挿入します。

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY( 5000 ) buffer;
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE VARIABLE hold_blob LONG BINARY;
EXEC SQL SET hold_blob = '';
for(;;) {
  /* read some data into buffer ... */
  size = fread( &(buffer.array), 1, 5000, fp );
  if( size <= 0 ) break;
  buffer.len = size;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    /* add data to blob using concatenation */
    EXEC SQL SET hold_blob = hold_blob || :buffer;
  }
  EXEC SQL INSERT INTO some_table VALUES ( 1, hold_blob );
  EXEC SQL DROP VARIABLE hold_blob;
```

SET 文 [T-SQL]

Adaptive Server Enterprise との互換性を保つ方法で現在の接続にデータベースオプションを設定します。

構文

SET *option-name option-value*

備考

使用可能なオプションは以下のとおりです。

オプション名	オプション値
ansinull	On または Off
ansi_permissions	On または Off
close_on_endtrans	On または Off
datefirst	1、2、3、4、5、6、または 7 このオプションの設定は、DATEPART 関数で weekday 値を取得するときに影響を与えます。
quoted_identifier	On または Off
rowcount	*integer*
self_recursion	On または Off
string_rtruncation	On または Off
textsize	*integer*
トランザクションの独立性レベル	0、1、2、3、snapshot、statement snapshot、または read only statement snapshot

SQL Anywhere のデータベースオプションは、SET OPTION 文を使用して設定されます。ただし、互換性を確保するため、便利なオプションについては SQL Anywhere でも Adaptive Server Enterprise の SET 文がサポートされています。

次のオプションは、SQL Anywhere と Adaptive Server Enterprise で Transact-SQL SET 文を使用して設定できます。

- **SET ansinull { On | Off }**　SQL Anywhere と Adaptive Server Enterprise では、値を NULL と比較するデフォルトの動作が異なります。ansinull を Off に設定する場合、NULL との比較に対して Transact-SQL との互換性が提供されます。

  SQL Anywhere では、次の構文もサポートされています。

  SET ansi_nulls { **On** | **Off** }

  詳細については、「ansinull オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

- **SET ansi_permissions { On | Off }**　カラム参照を含む UPDATE または DELETE を実行するのに必要なパーミッションに関する、SQL Anywhere と Adaptive Server Enterprise のデフォルト動作は異なります。ansi_permissions を Off に設定すると、UPDATE と DELETE のパーミッションに関して Transact-SQL との互換性が提供されます。

- **SET close_on_endtrans { On | Off }**　トランザクションの終わりでカーソルを閉じるデフォルト動作は、SQL Anywhere と Adaptive Server Enterprise で異なります。close_on_endtrans を Off に設定すると、Transact-SQL との互換性が提供されます。

- **SET datefirst { 1 | 2 | 3 | 4 | 5 | 6 | 7 }**　デフォルトは 7 です。これは、週の開始日が日曜日であることを意味します。

- **SET quoted_identifier { On | Off }**　二重引用符で囲まれた文字列を識別子 (On) として解釈するか、単なる文字列 (Off) として解釈するかを制御します。

- **SET rowcount**　*integer* Transact-SQL の ROWCOUNT オプションは、カーソルについてフェッチするローの数の上限を指定の整数に設定します。カーソルを再位置付けしてフェッチしたローにも適用されます。この上限を超えたフェッチには警告が発せられます。このオプション設定は OPEN の要求が出て、カーソルがフェッチするロー数の推測値が返されるときに考慮されます。

  また、SET ROWCOUNT では、検索する UPDATE 文または DELETE 文の対象となるローの数を *integer* に制限します。たとえば、この機能を使用すると、一定の間隔で COMMIT 文を実行し、ロールバックログとロックテーブルのサイズを制限することができます。アプリケーション (またはプロシージャー) は、最初の操作が影響するローに更新／削除を再発行させるループを指定する必要があります。次に簡単な例を示します。

```
BEGIN
  DECLARE @count INTEGER
  SET rowcount 20
  WHILE(1=1) BEGIN
    UPDATE 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
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    END
    SET rowcount 0
  END
```

SQL Anywhere では、ROWCOUNT 設定が Interactive SQL が表示できるローの数より大きい場合、Interactive SQL は追加のフェッチを行ってカーソルを再配置することができます。そのため、実際に表示されるローの数は、要求した数より少なくなることがあります。また、切り捨ての警告のために、ローが再度フェッチされる場合、カウントは正しくない場合があります。

値を 0 にすると、オプションをリセットし、すべてのローを取得します。

- **SET self_recursion { On | Off }**　self_recursion オプションをトリガー内で使用して、そのトリガーに関連するテーブルの操作が他のトリガーを起動できるか (On) できないか (Off) を設定します。

- **SET string_rtruncation { On | Off }**　SQL 文字列データ代入時にスペース以外の文字がトランケートされるときのデフォルト動作は、Adaptive Server Anywhere と Adaptive Server Enterprise では異なります。string_rtruncation を On に設定すると、文字列の比較に関して、Transact-SQL との互換性が提供されます。

- **SET textsize**　SELECT 文で返される TEXT 型データまたは IMAGE 型データの最大サイズをバイト単位で指定します。@@textsize グローバル変数は、現在の設定を格納します。デフォルトサイズ (32 KB) にリセットするには、次のコマンドを使用します。

  ```
  set textsize 0
  ```

- **SET transaction isolation level { 0 | 1 | 2 | 3 | snapshot | statement snapshot | readonly statement snapshot }**　現在の接続のロック独立性レベルを設定します。

  Adaptive Server Enterprise では、1 と 3 だけが有効なオプションです。SQL Anywhere の場合、0、1、2、3、snapshot、statement snapshot、read only statement snapshot が有効なオプションです。

SET 文は、Adaptive Server Anywhere において互換性のために prefetch オプションで使用できますが、効力はありません。

パーミッション

なし

関連する動作

なし

参照

- 「独立性レベルと一貫性」『SQL Anywhere サーバー SQL の使用法』
- 「Transact-SQL 互換性のオプション」『SQL Anywhere サーバー SQL の使用法』
- 「互換性オプション」『SQL Anywhere サーバー データベース管理』
- 「isolation_level オプション」『SQL Anywhere サーバー データベース管理』
- 「ansi_permissions オプション」『SQL Anywhere サーバー データベース管理』
- 「close_on_endtrans オプション」『SQL Anywhere サーバー データベース管理』
- 「quoted_identifier オプション」『SQL Anywhere サーバー データベース管理』
- 「string_rtruncation オプション」『SQL Anywhere サーバー データベース管理』
- 「first_day_of_week オプション」『SQL Anywhere サーバー データベース管理』
- 「DATEPART 関数 [日付と時刻]」199 ページ
- 「SET OPTION 文」913 ページ

標準と互換性

- **SQL/2008**　Transact-SQL 拡張。

SETUSER 文

データベース管理者が同じ接続で別の認証されたユーザーの ID を使用することを許可します。

構文

{ **SET SESSION AUTHORIZATION** | **SETUSER** }
[[**WITH OPTION**] *userid*]

パラメーター

WITH OPTION 句　デフォルトでは、パーミッション (グループメンバーシップを含む) だけが変更されます。WITH OPTION を指定すると、有効なデータベースオプションは *userid* の現在のデータベースオプションに変更されます。

userid　ユーザー ID は、識別子 (SETUSER 構文) または文字列 (SET SESSION AUTHORIZATION 構文) です。

備考

SETUSER 文は、管理作業のために提供されており、接続プーリングには使用しないでください。SETUSER 文を実行した後、以下のいずれかのコマンドを実行すると、使用しているユーザー認証を確認できます。

- SELECT USER
- SELECT CURRENT USER

ユーザー ID のない SETUSER は、前に実行したすべての SETUSER 文を元に戻します。

プロシージャー、トリガー、イベントハンドラー、またはバッチ内では SETUSER 文を使用できません。

SETUSER 文の使用法には、次のようなものがあります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **オブジェクトの作成**　SETUSER を使用して、別のユーザーが所有することになるデータベースオブジェクトを作成できます。
- **パーミッションのチェック**　データベース管理者は、別のユーザーのパーミッションとグループメンバーシップを使用し、クエリ、プロシージャー、ビューなどのパーミッションや名前の決定をテストできます。
- **管理者に安全な環境を提供**　データベース管理者はデータベースでのすべての動作を行うパーミッションを持ちます。不注意から意図しない動作を行うことを確実に回避したい場合は、SETUSER を使用してパーミッションの少ない別のユーザー ID に切り替えることができます。

注意
プロシージャー、トリガー、イベント、またはバッチ内では SETUSER 文を使用できません。

パーミッション

DBA 権限。

参照

- 「EXECUTE IMMEDIATE 文 [SP]」740 ページ
- 「GRANT 文」782 ページ
- 「REVOKE 文」887 ページ
- 「SET OPTION 文」913 ページ
- 「識別子」4 ページ
- 「文字列」5 ページ

標準と互換性

- **SQL/2008**　SET SESSION AUTHORIZATION 構文は、SQL/2008 のオプションの言語機能 F321、"User authorization" の一部です。SETUSER 構文はベンダー拡張です。WITH OPTION 構文はいずれの変形でも使用できますが、WITH OPTION はベンダー拡張です。

例

次の文は、ユーザー名 DBA によって実行され、ユーザー ID を Joe に変更し、さらに Jane に変更した後、DBA に戻します。

```
SETUSER "Joe"
// ... operations...
SETUSER WITH OPTION "Jane"
// ... operations...
SETUSER
```

次の文は、ユーザーを Jane に設定します。ユーザー ID は、識別子ではなく文字列として提供されます。

```
SET SESSION AUTHORIZATION 'Jane';
```

SIGNAL 文 [SP]

例外条件を通知します。

構文

SIGNAL *exception-name*

備考

SIGNAL を使うと、例外を起こすことができます。

exception-name は、現在の複合文の始めにある DECLARE 文を使用して宣言された例外の名前を指定する場合に使用します。この例外は、システム定義の SQLSTATE またはユーザー定義の SQLSTATE と合致していなければなりません。ユーザー定義 SQLSTATE の値は 99000 ～ 99999 の範囲になければなりません。

パーミッション

なし

関連する動作

なし

参照

- 「RESIGNAL 文 [SP]」879 ページ
- 「BEGIN 文」497 ページ
- 「例外ハンドラー」『SQL Anywhere サーバー SQL の使用法』
- 「RAISERROR 文」862 ページ

標準と互換性

- **SQL/2008**　SIGNAL 文は、SQL/2008 のオプションの言語機能 P002、"Computational completeness" の一部です。

- **Transact-SQL**　SIGNAL 文は、Transact-SQL 複合文およびプロシージャーでは使用できません。

例

次の複合文は、ユーザー定義の例外を宣言し、その信号を送ります。この例を Interactive SQL から実行すると、**[結果]** 領域の **[メッセージ]** タブにメッセージ **My exception signaled** が表示されます。

```
BEGIN
  DECLARE myexception EXCEPTION
  FOR SQLSTATE '99001';
  SIGNAL myexception;
  EXCEPTION
    WHEN myexception THEN
      MESSAGE 'My exception signaled'
      TO CLIENT;
END
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

START DATABASE 文

データベースを現在のデータベースサーバーで起動します。

構文

START DATABASE *database-file* [*start-options* ...]

start-options :
[**AS** *database-name*]
[**ON** *database-server-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** }]

パラメーター

database-file *database-file* パラメーターは文字列です。*database-file* に相対パスを指定する場合、そのパスはデータベースサーバーの開始ディレクトリを基準にします。

***start-options* 句** *start-options* 句は任意の順序で表示できます。

○ **AS 句** *database-name* を指定しない場合は、デフォルト名がデータベースに割り当てられます。このデフォルト名は、データベースファイルのルートです。たとえば、ファイル *C:¥Database Files¥demo.db* 内のデータベースにはデフォルト名の demo が付きます。*database-name* パラメーターは識別子です。

○ **ON 句** この句は Interactive SQL でのみサポートされます。Interactive SQL では、*server-name* を指定しない場合、デフォルトサーバーは、現在実行しているサーバーのうち、最初に起動したサーバーです。*server-name* パラメーターは識別子です。

○ **WITH TRUNCATE AT CHECKPOINT 句** チェックポイントのログの切り捨てを有効にしてデータベースを起動します。

○ **FOR READ ONLY 句** 読み込み専用モードでデータベースを起動します。リカバリが必要なデータベースで使用した場合は文が失敗し、エラーが返されます。

○ **AUTOSTOP 句** AUTOSTOP 句のデフォルト設定は ON です。AUTOSTOP を ON に設定すると、最後の接続が削除されるときに、データベースが自動的にアンロードされます。AUTOSTOP を OFF に設定すると、データベースはアンロードされません。

Interactive SQL では、ON と OFF の代わりに YES と NO も使用できます。

○ **KEY 句** データベースが強力に暗号化されている場合は、この句を使用して KEY 値 (パスワード) を入力します。

○ **WITH SERVER NAME 句** この句は、データベースサーバーに接続するときにそのサーバーの代替名を指定するときに使用します。

注意
データベースミラーリングを使用している場合は、プライマリサーバーとミラーサーバーがそれぞれどちらのサーバーなのかを事前に把握していなくてもクライアントアプリケーションが現在のプライマリサーバーに接続できるように、代替サーバー名を指定する必要があります。両方のパートナーサーバーでは、代替サーバー名として同じ名前を使用することが必要です。CREATE MIRROR SERVER 文を使用すると、ミラーリングシステムのプライマリサーバーとミラーサーバーの代替サーバー名を作成できます。

○ **DIRECTORY 句**　この句を使用して、起動しようとしているデータベースの DB 領域が配置されているディレクトリを指定します。たとえば、データベースサーバーが DB 領域と同じディレクトリで起動されている場合に DIRECTORY '.' 句があると、データベースサーバーに対して現在のディレクトリにあるすべての DB 領域を検出するように指示することになります。

○ **CHECKSUM 句**　この句は、グローバルチェックサムを有効にして作成されていないデータベースに新しく書き込まれるページに対して、書き込みチェックサムを有効にするときに使用します。この句を指定すると、-wc データベースオプションと同じ動作になります。

グローバルチェックサムを有効にしてデータベースを作成することと CHECKSUM 句の違いは、CHECKSUM ON を指定した場合、ディスクに書き込まれるときにのみ、データベースページにチェックサムが追加されることです。ディスクから読み込まれるページは、チェックサム値を計算してからページが書き込まれたかどうかを検証されるのみです。データベースでグローバルチェックサムが有効にされている場合、書き込み時にすべてのページに対してチェックサムが計算され、読み込み時にすべてのページに対してチェックサムが検証されます。

データベースサーバーが Windows Mobile またはリムーバブルストレージデバイス (ネットワーク共有または USB デバイスなど) 上でデータベースが実行されていることを検出した場合、データベースサーバーはすべてのデータベースページに対して、書き込みチェックサムを自動的に有効にします。

デフォルトでは、SQL Anywhere バージョン 10 および 11 で作成されたデータベースは、グローバルチェックサムが有効にされていません。バージョン 12 以降のデータベースサーバーで、SQL Anywhere 10 または 11 で作成されたデータベースを起動すると、ページがディスクに書き込まれるときに、デフォルトでデータベースサーバーは、ページに対する書き込みチェックサムを作成します (CHECKSUM ON)。バージョン 12 以降のデータベースは、デフォルトでグローバルチェックサムが有効にされており、デフォルトですべてのデータベースページにチェックサムがあるため、データベースサーバーはこれらのデータベースに対して CHECKSUM OFF をデフォルト設定します。デフォルトのチェックサム設定を使用しない場合、-wc オプションまたは START DATABASE 文を使用して、データベースサーバーのチェックサムの動作を変更できます。

次の文を実行することによって、データベースがグローバルチェックサムを有効にして作成されたかどうかをチェックできます。

```
SELECT DB_PROPERTY ( 'Checksum' );
```

次の文を実行すると、書き込みチェックサムが有効にされているかどうかをチェックできます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT DB_PROPERTY ( 'WriteChecksum' );
```

備考

指定したデータベースを現在のデータベースサーバーで起動します。

データベースに接続していない状態で、START DATABASE 文を使用する場合、まずユーティリティデータベースなどのデータベースに接続します。

START DATABASE 文は、指定されたデータベースに現在のアプリケーションを接続しません。START DATABASE 文を使用する場合も明示的な接続が必要です。

Interactive SQL は ON 句をサポートします。この句によって、データベースを現在のデータベースサーバー以外のサーバーで起動できます。

SQL Anywhere ユーティリティデータベースへの接続に使用できるのは utility_db という名前のデータベースのみです。

パーミッション

必要なパーミッションは、データベースサーバーの -gd オプションで指定します。このオプションは、パーソナルデータベースサーバーでは all、ネットワークサーバーでは DBA にデフォルトで設定されます。

関連する動作

なし

参照

- 「STOP DATABASE 文」946 ページ
- 「CREATE MIRROR SERVER 文」579 ページ
- 「CONNECT 文 [ESQL] [Interactive SQL]」516 ページ
- 「VALIDATE 文」983 ページ
- 「-gd dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「-ds dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「-ek dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「-m dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「-r dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「-n dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「-wc dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「-sn dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリングシステムの設定」『SQL Anywhere サーバー データベース管理』
- 「ユーティリティデータベース」『SQL Anywhere サーバー データベース管理』
- 「チェックサムを使用した破損の検出」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

現在のサーバーでデータベースファイル *C:¥Database Files¥sample_2.db* を起動します。

```
START DATABASE 'c:¥database files¥sample_2.db';
```

Interactive SQL から、データベースファイル *c:¥Database Files¥sample_2.db* を sam2 としてサーバー sample 上で起動します。

```
START DATABASE 'c:¥database files¥sample_2.db'
AS sam2
ON sample;
```

START SERVER 文 [Interactive SQL]

データベースサーバーを起動します。

構文

START SERVER AS *database-server-name* [**STARTLINE** *command-string*]

備考

START SERVER 文は、データベースサーバーを起動します。データベースサーバーに対してオプションのセットを指定する場合、STARTLINE キーワードをコマンド文字列と一緒に使います。

START ENGINE は互換性のために受け入れられますが、推奨されません。

パーミッション

なし

関連する動作

なし

参照

- 「STOP SERVER 文」949 ページ
- 「SQL Anywhere データベースサーバーの構文」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

sample という名前のデータベースサーバーを、1 つもデータベースを起動しない状態で起動します。

```
START SERVER AS sample;
```

次の例は、STARTLINE 句の使い方を示します。

```
START SERVER AS eng1 STARTLINE 'dbsrv12 -c 8M';
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

START EXTERNAL ENVIRONMENT 文

外部環境を開始します。

構文

START EXTERNAL ENVIRONMENT *environment-name*

environment-name :
JAVA
| **PERL**
| **PHP**
| **CLR**
| **C_ESQL32**
| **C_ESQL64**
| **C_ODBC32**
| **C_ODBC64**

パラメーター

environment-name 開始する外部環境の名前。

備考

外部環境の詳細については、「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』を参照してください。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』
- 「ALTER EXTERNAL ENVIRONMENT 文」420 ページ
- 「STOP EXTERNAL ENVIRONMENT 文」947 ページ
- 「INSTALL EXTERNAL OBJECT 文」807 ページ
- 「REMOVE EXTERNAL OBJECT 文」875 ページ
- 「SYSEXTERNENV システムビュー」1265 ページ
- 「SYSEXTERNENVOBJECT システムビュー」1266 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

Perl 外部環境を開始します。

```
START EXTERNAL ENVIRONMENT PERL;
```

START JAVA 文

Java VM を起動します。

構文

START JAVA

備考

START JAVA 文は、Java VM を起動します。主な使い方として、Java VM を適宜ロードし、ユーザーが Java の機能を使い始めるときに、Java VM ロード中の一時停止が発生しないようにします。

Java VM の場所を特定できるようにデータベースサーバーを設定します。データベースごとに異なる Java VM を指定できるため、java_location オプションを使用して Java VM の場所 (パス) を指定できます。

パーミッション

Java VM をインストールし、データベースを Java 実行可能にしてください。

この文は、Windows Mobile ではサポートされません。

関連する動作

なし

参照

- 「STOP JAVA 文」948 ページ
- 「Java VM の起動と停止」『SQL Anywhere サーバー プログラミング』
- 「java_location オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

Java VM を起動します。

```
START JAVA;
```

START LOGGING 文 [Interactive SQL]

実行された SQL 文のログファイルへのロギングを開始します。

構文

START LOGGING *filename*

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

START LOGGING 文は、それ以降に実行されるすべての SQL 文の指定されたログファイルへのコピーを開始します。このファイルが存在しない場合は、Interactive SQL によって作成されます。ロギングは、STOP LOGGING 文で明示的に停止するか、現在の Interactive SQL セッションを終了するまで継続します。

[SQL] » [ロギングの開始] と **[SQL] » [ロギングの停止]** をクリックして、ロギングを開始したり停止したりすることもできます。

ロギングと実行時間レポートの両方が有効である場合は、実行時間がログファイルに記録されます。

パーミッション

なし

関連する動作

なし

参照

- 「STOP LOGGING 文 [Interactive SQL]」949 ページ
- 「isql_command_timing オプション [Interactive SQL]」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL での文のロギング」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

c: ディレクトリにあるファイル *filename.sql* へのロギングを開始します。

```
START LOGGING 'c:¥filename.sql';
```

START SUBSCRIPTION 文 [SQL Remote]

パブリケーションに対するユーザーのサブスクリプションを開始します。

構文

START SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*, ...

パラメーター

publication-name　このパブリケーション名でユーザーのサブスクリプションが作成されます。パブリケーションの所有者を含めることもできます。

subscription-value パブリケーションのサブスクリプション式と照合される文字列。サブスクライバーは単一のパブリケーションに対して複数のサブスクリプションを持つことがあるので、この値は必須です。

subscriber-id パブリケーションに対するサブスクライバーのユーザー ID。このユーザーは、パブリケーションへのサブスクリプションを持っている必要があります。

備考

SQL Remote サブスクリプションは、パブリケーションの更新が統合データベースからリモートデータベースへ送信されたときに開始したと見なされます。

START SUBSCRIPTION 文は、サブスクリプションを管理する一連の文の 1 つです。CREATE SUBSCRIPTION 文は、サブスクライバーが受信するデータを定義します。SYNCHRONIZE SUBSCRIPTION 文は、統合データベースとリモートデータベース間の一貫性を確保するためのものです。メッセージのサブスクライバーへの送信を開始するには、START SUBSCRIPTION 文が必ず必要です。

各サブスクリプションの最後のデータは、サブスクリプションを開始する前のデータと一致していなければなりません。データベース抽出ユーティリティを使用してサブスクリプションの作成、同期、起動を管理することをおすすめします。データベース抽出ユーティリティを使用すれば明示的に START SUBSCRIPTION 文を実行する必要はなくなります。また、Message Agent も、サブスクリプションが同期されるとサブスクリプションを起動します。

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「CREATE SUBSCRIPTION 文 [SQL Remote]」645 ページ
- 「REMOTE RESET 文 [SQL Remote]」874 ページ
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」958 ページ
- 「STOP SUBSCRIPTION 文 [SQL Remote]」950 ページ
- 「抽出ユーティリティ (dbxtract)」『SQL Remote』
- 「サブスクリプションの開始」『SQL Remote』
- 「複数のリモートデータベースの作成」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、パブリケーション pub_contact に対するユーザー SamS のサブスクリプションを起動します。

```
START SUBSCRIPTION TO pub_contact
FOR SamS;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

START SYNCHRONIZATION DELETE 文 [Mobile Link]

Mobile Link 同期で行われた削除のロギングを再起動します。

構文

START SYNCHRONIZATION DELETE

備考

通常、SQL Anywhere と Ultra Light は、同期の一部であるテーブルやカラムに対して行われたすべての変更のログを自動的に取り、次の同期中にこれらの変更を統合データベースにアップロードします。STOP SYNCHRONIZATION DELETE 文を使用すると、削除処理の自動ロギングを一時的に中断できます。START SYNCHRONIZATION DELETE 文を使用すると、自動ロギングを再起動できます。

STOP SYNCHRONIZATION DELETE 文を実行すると、その接続に対してそれ以降に実施された削除操作は同期されません。この効果は、START SYNCHRONIZATION DELETE 文が実行されるまで継続します。STOP SYNCHRONIZATION DELETE を繰り返してもそれ以上効果はありません。

STOP SYNCHRONIZATION DELETE 文の実行回数にかかわらず、START SYNCHRONIZATION DELETE 文を 1 回実行すれば、ロギングが再起動します。

アプリケーションでデータを同期しない場合は、START SYNCHRONIZATION DELETE を使用しないでください。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」951 ページ
- ULConnection.StartSynchronizationDelete メソッド [Ultra Light.NET]『Ultra Light .NET プログラミング』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の一連の SQL 文は、START SYNCHRONIZATION DELETE と STOP SYNCHRONIZATION DELETE の使用方法を示します。

```
-- 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;
```

START SYNCHRONIZATION SCHEMA CHANGE 文 [Mobile Link]

Mobile Link 同期スキーマの変更を開始します。

構文

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*

subscription-name : *identifier*

パラメーター

FOR TABLES 句　この句は、スキーマの変更によって影響を受けるテーブルを指定します。

SET SCRIPT VERSION 句　FOR TABLES 句で指定されたテーブルを含むすべてのサブスクリプションの新しいスクリプトバージョンを指定します。新しいスクリプトバージョンは、既存のスクリプトバージョンと同じでもかまいません。

「クラウドに関する注意：」クラウドテナントデータベースの *script-version* 値に **NULL** を指定できます。

ON SUBSCRIPTION 句　指定されたサブスクリプションに新しいスクリプトバージョンを指定します。この句を使用する場合、FOR TABLES 句で指定されたテーブルを含むサブスクリプ

ションごとに、この句を繰り返す必要があります。新しいスクリプトバージョンは、既存のスクリプトバージョンと同じでもかまいません。

備考

スキーマの変更を適用するすべてのテーブルを *table-list* にリストしてください。テーブルを複数回リストすることはできません。*table-list* に指定されているテーブルに既存のロックがある場合、エラーメッセージが表示されます。

データベースで一度に実行できる同期スキーマの変更は 1 つのみです。別のスキーマの変更が処理中の場合、START SYNCHRONIZATION SCHEMA CHANGE 文は失敗します。

データベースサーバーは、*table-list* に指定されているすべてのテーブルのロックを取得します。データベースサーバーは、ロックを取得するとき、ブロックオプションの設定を無視します。ロックを取得できない場合、それまでに取得したすべてのロックが解放され、エラーメッセージが表示されます。

同期スキーマの変更中は、

- データ修正文を実行できません。
- 別の START SYNCHRONIZATION SCHEMA CHANGE 文を実行できません。
- パブリケーションを変更して、*table-list* にあるテーブルのカラムのサブセット方法を変更できます。
- パブリケーションを変更して、*table-list* にあるテーブルを削除できます。
- *table-list* にリストされている任意のテーブルを変更できます。

START SYNCHRONIZATION SCHEMA CHANGE 文の実行の前後に、暗黙的なコミットが実行されます。同期スキーマの変更は、STOP SYNCHRONIZATION SCHEMA CHANGE 文によって終了します。STOP SYNCHRONIZATION SCHEMA CHANGE 文が実行されると、テーブルロックがすべて解放されます。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「STOP SYNCHRONIZATION SCHEMA CHANGE 文 [Mobile Link]」957 ページ
- SynchronizationSchemaChangeActive データベースプロパティ『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の一連の SQL 文は、START SYNCHRONIZATION SCHEMA CHANGE と STOP SYNCHRONIZATION SCHEMA CHANGE の使用方法を示します。

```
START SYNCHRONIZATION SCHEMA CHANGE
  FOR TABLES DBA.Sales, DBA.Products
  SET SCRIPT VERSION = 'version_2';
ALTER TABLE DBA.Sales ADD SUBTOTAL NUMERIC (10,2);
ALTER TABLE DBA.Products ALTER QUANTITY BIGINT;
STOP SYNCHRONIZATION SCHEMA CHANGE;
```

STOP DATABASE 文

現在のデータベースサーバー上のデータベースを停止します。

構文

STOP DATABASE *database-name*
[**ON** *database-server-name*]
[**UNCONDITIONALLY**]

パラメーター

STOP DATABASE 句　*database-name* は、現在のサーバーで動作している (現在のデータベース以外の) データベースの名前です。

ON 句　この句は Interactive SQL でのみサポートされます。Interactive SQL で *database-server-name* を指定しないと、すべての実行しているサーバーについて、指定したデータベースを検索します。

Interactive SQL でこの文を使用しない場合、データベースは現在のデータベースサーバーで起動された場合のみ停止されます。

UNCONDITIONALLY 句　データベースとの接続がある場合でも、そのデータベースを停止します。デフォルトでは、接続があるとデータベースは停止しません。

備考

STOP DATABASE 文は、現在のデータベースサーバー上の指定したデータベースを停止します。

パーミッション

必要なパーミッションは、データベースサーバーの -gd オプションで指定します。このオプションは、パーソナルデータベースサーバーでは all、ネットワークサーバーでは DBA にデフォルトで設定されます。

現在接続しているデータベースに対して STOP DATABASE を使用することはできません。

関連する動作

なし

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「START DATABASE 文」935 ページ
- 「DISCONNECT 文 [ESQL] [Interactive SQL]」707 ページ
- 「-gd dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「サーバー停止ユーティリティ (dbstop)」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

現在のサーバー上のデータベース *sample* を停止します。

```
STOP DATABASE sample;
```

STOP EXTERNAL ENVIRONMENT 文

外部環境を停止します。

構文

STOP EXTERNAL ENVIRONMENT *environment-name*

environment-name :
JAVA
| **PERL**
| **PHP**
| **CLR**
| **C_ESQL32**
| **C_ESQL64**
| **C_ODBC32**
| **C_ODBC64**

パラメーター

environment-name　停止する外部環境の名前。

備考

外部環境の詳細については、「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』を参照してください。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「SQL Anywhere 外部環境のサポート」『SQL Anywhere サーバー プログラミング』
- 「ALTER EXTERNAL ENVIRONMENT 文」420 ページ
- 「START EXTERNAL ENVIRONMENT 文」939 ページ
- 「INSTALL EXTERNAL OBJECT 文」807 ページ
- 「REMOVE EXTERNAL OBJECT 文」875 ページ
- 「SYSEXTERNENV システムビュー」1265 ページ
- 「SYSEXTERNENVOBJECT システムビュー」1266 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

Perl 外部環境を停止します。

```
STOP EXTERNAL ENVIRONMENT PERL;
```

STOP JAVA 文

Java VM を停止します。

構文

STOP JAVA

備考

STOP JAVA 文は、使用されていない Java VM をアンロードします。主な目的は、システムリソースを経済的に使用することです。

パーミッション

この文は、Windows Mobile ではサポートされません。

関連する動作

なし

参照

- 「START JAVA 文」940 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

Java VM を停止します。

```
STOP JAVA;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

STOP LOGGING 文 [Interactive SQL]

現在のセッションで実行される SQL 文のロギングを停止します。

構文

STOP LOGGING

備考

STOP LOGGING 文は、実行した各 SQL 文のログファイルへの書き込みを停止します。ロギングを開始するには、START LOGGING 文を使用します。

[SQL] » **[ロギングの停止]** をクリックしてロギングを停止することもできます。

パーミッション

なし

関連する動作

なし

参照

- 「START LOGGING 文 [Interactive SQL]」940 ページ
- 「Interactive SQL での文のロギング」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は、現在のロギングセッションを停止します。

```
STOP LOGGING;
```

STOP SERVER 文

データベースサーバーを停止します。

構文

STOP SERVER [*database-server-name*] [**UNCONDITIONALLY**]

パラメーター

UNCONDITIONALLY 句　停止するデータベースサーバーに接続しているユーザーが他にいない場合、UNCONDITIONALLY を使用する必要はありません。他にも接続しているユーザーがいる場合は、UNCONDITIONALLY キーワードを指定した場合のみデータベースサーバーが停止します。

備考

database-server-name は Interactive SQL でのみ使用できます。この文を Interactive SQL から実行しなかった場合、現在のデータベースサーバーが停止されます。

デフォルトでは、データベースサーバーは他の接続がある場合には停止しません。UNCONDITIONALLY 句を使用すると、データベースサーバーはデータベースサーバーへの他の接続がある場合でも停止します。STOP SERVER 文をクライアント接続で実行した場合、サーバーが正常に停止すると、通信エラーが発生します。

STOP SERVER 文は、ストアドプロシージャー、トリガー、イベント、またはバッチでは使用できません。

互換性を保つために STOP ENGINE はまだ使用できますが、廃止される予定です。

パーミッション

サーバーを停止するために必要なパーミッションは、データベースサーバーコマンドラインの -gk 設定によって指定されます。パーソナルサーバーのデフォルト設定は all で、ネットワークサーバーのデフォルト設定は DBA です。

関連する動作

なし

参照

- 「START SERVER 文 [Interactive SQL]」938 ページ
- 「-gk dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

現在のデータベースサーバーを他に接続がない場合のみ停止します。

```
STOP SERVER;
```

STOP SUBSCRIPTION 文 [SQL Remote]

パブリケーションに対するユーザーのサブスクリプションを停止します。

構文

STOP SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*, ...

パラメーター

publication-name　このパブリケーション名でユーザーのサブスクリプションが作成されます。パブリケーションの所有者を含めることもできます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

subscription-value パブリケーションのサブスクリプション式と照合される文字列。サブスクライバーは単一のパブリケーションに対して複数のサブスクリプションを持つことがあるので、この値は必須です。

subscriber-id パブリケーションに対するサブスクライバーのユーザー ID。このユーザーは、パブリケーションへのサブスクリプションを持っている必要があります。

備考

SQL Remote サブスクリプションは、パブリケーションの更新が統合データベースからリモートデータベースへ送信されたときに開始したと見なされます。

サブスクライバーに送るメッセージを停止するには、STOP SUBSCRIPTION 文が必要です。サブスクライバーに送るメッセージを再開するには、START SUBSCRIPTION 文が必要です。ただし、サブスクリプションを再開する場合は足りないメッセージがないように、サブスクリプションが正しく同期していることを確認してください。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「DROP SUBSCRIPTION 文 [SQL Remote]」 728 ページ
- 「START SUBSCRIPTION 文 [SQL Remote]」 941 ページ
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」 958 ページ
- 「抽出ユーティリティ (dbxtract)」『SQL Remote』
- 「サブスクリプションの停止」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の文は、パブリケーション pub_contact に対するユーザー SamS のサブスクリプションを停止します。

```
STOP SUBSCRIPTION TO pub_contact
FOR SamS;
```

STOP SYNCHRONIZATION DELETE 文 [Mobile Link]

Mobile Link 同期で行われた削除のロギングを一時的に停止します。

構文

STOP SYNCHRONIZATION DELETE

備考

通常、SQL Anywhere と Ultra Light リモートデータベースは、同期中のテーブルやカラムに対して行われたすべての変更のログを自動的に取り、次の同期中にこれらの変更を統合データベースにアップロードします。この文を使用して、SQL Anywhere または Ultra Light リモートデータベースに対する削除操作のロギングを一時的に中断できます。

接続が STOP SYNCHRONIZATION DELETE を実行するときと START SYNCHRONIZATION DELETE を実行するときの間に、その接続で実行された削除操作は同期されません。

STOP SYNCHRONIZATION DELETE 文の実行回数にかかわらず、START SYNCHRONIZATION DELETE 文を 1 回実行すれば、ロギングが再起動します。

この文は、リモートデータベースに対して訂正を行うには便利ですが、Mobile Link 同期を事実上無効にしてしまうので、使用の際には注意してください。

アプリケーションでデータを同期しない場合は、STOP SYNCHRONIZATION DELETE を使用しないでください。

パーミッション

DBA 権限

関連する動作

なし

参照

- ULConnection.StartSynchronizationDelete メソッド [Ultra Light.NET]『Ultra Light .NET プログラミング』
- ULConnection.StopSynchronizationDelete メソッド [Ultra Light.NET]『Ultra Light .NET プログラミング』
- 「START SYNCHRONIZATION DELETE 文 [Mobile Link]」943 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

例については、「START SYNCHRONIZATION DELETE 文 [Mobile Link]」943 ページを参照してください。

SYNCHRONIZE 文 [Mobile Link]

SQL Anywhere データベースを Mobile Link サーバーと同期します。文自体に同期オプションを指定できます。

構文

SYNCHRONIZE {
PROFILE *sync-profile-name* [**MERGE** *sync-option* [;...]]

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| **USING** *sync-option* [;...]
| **START**
| **STOP**
}

[**PORT** *port-number*]
[**VERBOSITY** { **LOW** | **NORMAL** | **HIGH** }]
[**TIMEOUT** *timeout*]
[**USER** *user-name* **IDENTIFIED BY** *password*]

sync-option : *string*

パラメーター

sync-profile-name この同期に使用する同期プロファイルの名前。

MERGE 句 この句は、同期プロファイルオプションを追加または上書きするために使用します。

USING 句 この句は、同期プロファイルを使用しないで同期プロファイルオプションを指定するために使用します。

sync-option 1 つ以上の同期プロファイルオプションの値ペアをセミコロンで区切った文字列です。たとえば、'option1=value1;option2=value2' のように記述します。

PORT 句 この句は、データベースサーバーが dbmlsync ユーティリティと通信するために使用するポート番号を指定するために使用します。デフォルトは 4433 です。

VERBOSITY 句 この句では、同期中に synchronize_results および synchronize_parameters 共有グローバルテンポラリテーブルに追加される情報量を指定します。

次に、各 VERBOSITY オプションで返されるクライアント API イベントのリストを示します。

オプション	返されるイベント
LOW	○ DBSC_EVENTTYPE_SYNC_START ○ DBSC_EVENTTYPE_SYNC_DONE ○ DBSC_EVENTTYPE_ERROR_MSG ○ DBSC_EVENTTYPE_WARNING_MSG
NORMAL (デフォルト)	○ DBSC_EVENTTYPE_SYNC_START ○ DBSC_EVENTTYPE_SYNC_DONE ○ DBSC_EVENTTYPE_ERROR_MSG ○ DBSC_EVENTTYPE_WARNING_MSG ○ DBSC_EVENTTYPE_INFO_MSG

オプション	返されるイベント
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

SYNCHRONIZE 文の VERBOSITY 句と、同期プロファイルに指定できる VERBOSITY オプションとを混同しないように注意してください。SYNCHRONIZE 文の VERBOSITY 句は、synchronize_results および synchronize_parameters テーブルに記録されるイベントの型を制御します。同期プロファイルの VERBOSITY オプションは、同期中に生成される DBSC_EVENTTYPE_INFO_MSG イベントの数を制御します。

```
SYNCHRONIZE PROFILE SalesData VERBOSITY NORMAL;
```

```
SYNCHRONIZE PROFILE SalesData MERGE 'Verbosity=BASIC,ROW_DATA' VERBOSITY NORMAL;
```

TIMEOUT 句　この句では、同期のキャンセルを試行するまでにデータベースサーバーが同期の完了を待機する時間を秒数で指定します。デフォルトは 240 秒です。

USER / IDENTIFIED BY 句　この句は、dbmlsync ユーティリティでデータベースを同期するために使用するデータベースユーザー ID とパスワードを指定するときに使用します。指定されるユーザー ID には、REMOTE DBA 権限または DBA 権限が必要です。デフォルトでは、同期には SYNCHRONIZE 文を実行したデータベース接続のユーザー ID が使用されます。

START 句　dbmlsync ユーティリティのサーバーモードでの実行を開始し、実行したままにします。同期は実行されません。短期間に複数の同期を実行している場合、この句を使用して、明示的に dbmlsync サーバーを起動して同期を実行し、STOP 句を使用して明示的に dbmlsync サーバーを停止することにより、パフォーマンスを向上させることができます。

STOP 句　以前に START 句を使用して起動した dbmlsync サーバーを停止します。同期は実行されません。

備考

この文は、Dbmlsync C++ API を含む SQL Anywhere 用 Mobile Link クライアントがインストールされている場合にのみ使用できます。

データベースサーバーを実行できないすべてのプラットフォームで、 SQL Anywhere 用 Mobile Link クライアントは利用できません。サポートされるプラットフォームのリストについては、http://www.sybase.com/detail?id=1061806 を参照してください。

同期が完了したら、synchronize_results と synchronize_parameters 共有グローバルテンポラリテーブルに格納されている同期の結果を表示できます。synchronize_results と synchronize_parameters テーブルには、データベースが起動されて以降、SYNCHRONIZE 文で実行されたすべての同期の結果が格納されます。synchronize_results と synchronize_parameters テーブルは、データベースサーバーがシャットダウンするたびにトランケートされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

synchronize_results テーブルには次のカラムが含まれます。

カラム名	データ型	説明
row_id	UNSIGNED BIGINT	ローがテーブルに挿入された順序を判断するために使用するテーブルのプライマリキー。
conn_id	UNSIGNED INT	イベントが synchronize_results テーブルに追加された時刻。
result_time	TIMESTAMP	イベントが synchronize_results テーブルに追加された時刻。
result_type	CHAR(128)	イベントの種類。

synchronize_results テーブルに表示される各イベントには、イベントの追加情報を含む関連パラメーターが存在することがあります。このパラメーターは、次のカラムを含む synchronize_parameters テーブルに格納されます。

カラム名	データ型	説明
row_id	UNSIGNED BIGINT	synchronize_results テーブルにある row_id カラムの外部キー。 この値を使用して、各パラメーターを属しているイベントに合わせる。
parm_id	UNSIGNED INT	パラメーターの 数値 ID を含める。 1 つ以上のパラメーターを持つイベントの場合は、この値を使用して、必要なパラメータの値を指定する。
parm_message	LONG VARCHAR	パラメーターに関する値。

過去または現在の同期の情報を表示するには、synchronize_results および synchronize_parameters テーブルに対して直接クエリを実行する代わりに、sp_get_last_synchronize_result システムプロシージャーを使用できます。

または、次の文を使用して、データベースサーバーの起動後に行われたすべての同期の結果を表示できます。

```
SELECT *
    FROM synchronize_results sr
```

```
KEY JOIN synchronize_parameters sp
ORDER BY sr.row_id , sp.parm_id
```

synchronize_results と synchronize_parameters テーブルを使用すると、現在の接続とは別の接続に対する同期の進行状況をモニターできます。別の接続に対する同期の進行状況をモニターするには、次の手順に従います。

●SELECT CONNECTION_PROPERTY 文を実行して、現在の接続の接続 ID を確認します。

●SYNCHRONIZE 文を実行して同期を開始します。

●別の接続で、use the sp_get_last_synchronize_results system プロシージャーを使用して、上で確認した接続 ID を使用した結果を取得します。

特定の接続で完了した同期または進行中の同期の結果を表示するには、sp_get_last_synchronize_results システムプロシージャーを使用します。

SYNCHRONIZE 文は、Ultra Light の SYNCHRONIZE 文に似ています。ただし、SQL Anywhere の SYNCHRONIZE 文は、dbmlsync ユーティリティをサーバーモードで起動して同期を実行します。Ultra Light の SYNCHRONIZE 文は、Ultra Light ランタイムを使用します。

データベースサーバーは Dbmlsync API クライアントとして機能し、TCP/IP を使用して dbmlsync サーバーと通信します。デフォルトでは、通信はポート 4433 で行われます。別のポートを指定するには、PORT 句を使用してください。

SYNCHRONIZE PROFILE 文と SYNCHRONIZE USING 文を使用して、同期を実行します。SYNCHRONIZE START と SYNCHRONIZE STOP 文を使用して、dbmlsync サーバーを起動または停止します。SYNCHRONIZE PROFILE 文または SYNCHRONIZE USING 文を実行すると、データベースサーバーはすでに実行中の dbmlsync サーバーへの接続を試みます。すでに実行中の dbmlsync サーバーが見つからない場合は、dbmlsync サーバーが起動します。同期が完了すると、データベースサーバーは起動した dbmlsync サーバーをシャットダウンします。すでに実行中の dbmlsync サーバーに文が接続した場合、dbmlsync サーバーはシャットダウンされません。複数の同期を実行しており、同期ごとに dbmlsync サーバーを起動または停止しないようにする場合は、SYNCHRONIZE START 文に続けて、複数の SYNCHRONIZE PROFILE 文または SYNCHRONIZE USING 文を実行し、最後に SYNCHRONIZE STOP 文を実行します。

パーミッション

REMOTE DBA 権限または DBA 権限

関連する動作

なし

参照

- 「CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link]」646 ページ
- Events and parm_id values from the synchronize_parameters table1215 ページ
- DBSC_Event 構造体 [Dbmlsync .NET]『Mobile Link クライアント管理』
- 「Mobile Link 同期」『Mobile Link クイックスタート』
- 「Mobile Link 同期プロファイル」『Mobile Link クライアント管理』
- 「SQL Anywhere Mobile Link クライアントの配備」『Mobile Link サーバー管理』
- 「sp_get_last_synchronize_result システムプロシージャー」1214 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、Test1 という同期プロファイルで同期するための構文を示します。

```
SYNCHRONIZE PROFILE Test1;
```

STOP SYNCHRONIZATION SCHEMA CHANGE 文 [Mobile Link]

Mobile Link の同期スキーマ変更を停止します。

構文

STOP SYNCHRONIZATION SCHEMA CHANGE

備考

STOP SYNCHRONIZATION SCHEMA CHANGE 文は、START SYNCHRONIZATION SCHEMA CHANGE 文で開始したスキーマ変更を停止します。START SYNCHRONIZATION SCHEMA CHANGE 文によって取得されるすべてのロックがリリースされます。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「START SYNCHRONIZATION SCHEMA CHANGE 文 [Mobile Link]」944 ページ
- 「SYSARTICLE システムビュー」1254 ページ
- SynchronizationSchemaChangeActive データベースプロパティ『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の一連の SQL 文は、START SYNCHRONIZATION SCHEMA CHANGE と STOP SYNCHRONIZATION SCHEMA CHANGE の使用方法を示します。

```
START SYNCHRONIZATION SCHEMA CHANGE
  ON DBA.Sales, DBA.Products
  SET SCRIPT VERSION = 'version 2';
ALTER TABLE DBA.Sales ADD SUBTOTAL NUMERIC (10,2);
ALTER TABLE DBA.Products ALTER QUANTITY BIGINT;
STOP SYNCHRONIZATION SCHEMA CHANGE;
```

SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]

パブリケーションに対するユーザーのサブスクリプションを同期します。

構文

SYNCHRONIZE SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *remote-user*, ...

パラメーター

publication-name このパブリケーション名でユーザーのサブスクリプションが作成されます。パブリケーションの所有者を含めることもできます。

subscription-value パブリケーションのサブスクリプション式と照合される文字列。サブスクライバーは単一のパブリケーションに対して複数のサブスクリプションを持つことがあるので、この値は必須です。

remote-user パブリケーションに対するサブスクライバーのユーザー ID。このユーザーは、パブリケーションへのサブスクリプションを持っている必要があります。

備考

SQL Remote サブスクリプションは、リモートデータベースのデータが統合データベースのデータと一致し、統合データベースからリモートデータベースにパブリケーションの更新を送信しても競合もエラーも発生しない状態の場合、「同期されている」と見なされます。

サブスクリプションを同期するには、統合データベース側のパブリケーションデータのコピーをリモートデータベースに送信します。SYNCHRONIZE SUBSCRIPTION 文は、これをメッセージシステムで自動的に実行します。できるかぎり、データベース抽出ユーティリティ (dbxtract) を使用して、メッセージシステムを使用せずにサブスクリプションを同期することをおすすめします。

パーミッション

DBA 権限

関連する動作

オートコミット。

参照

- 「CREATE SUBSCRIPTION 文 [SQL Remote]」645 ページ
- 「START SUBSCRIPTION 文 [SQL Remote]」941 ページ
- 「STOP SUBSCRIPTION 文 [SQL Remote]」950 ページ
- 「抽出ユーティリティ (dbxtract)」『SQL Remote』
- 「SQL Remote Message Agent (dbremote) を使用した同期」『SQL Remote』
- 「サブスクリプションの再同期」『SQL Remote』
- 「同期」『SQL Remote』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、パブリケーション pub_contact に対するユーザー SamS のサブスクリプションを同期します。

```
SYNCHRONIZE SUBSCRIPTION
  TO pub_contact
  FOR SamS;
```

SYSTEM 文 [Interactive SQL]

Interactive SQL から実行ファイルを起動します。

構文

SYSTEM '[*path* **]** *filename***'**

備考

指定した実行ファイルを起動します。パスとファイル名を一重引用符で囲んでください。

パーミッション

なし

関連する動作

なし

参照

- 「CONNECT 文 [ESQL] [Interactive SQL]」516 ページ
- 「Interactive SQL」『SQL Anywhere サーバー データベース管理』
- 「Interactive SQL ユーティリティ (dbisql)」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、メモ帳の実行ファイルが起動可能なパスにある場合に、メモ帳プログラムを起動します。

```
SYSTEM 'notepad.exe';
```

TRIGGER EVENT 文

指定したイベントをトリガーします。イベントは、イベントトリガーに対して定義したものでも、スケジュールされたイベントでもかまいません。

構文

TRIGGER EVENT *event-name* [(*parm* = *value*, ...)]

パラメーター

parm* = *value トリガー条件によってイベントハンドラーが実行されるときには、データベースサーバーが event_parameter 関数を使用して、イベントハンドラーにコンテキスト情報を渡すことができます。TRIGGER EVENT 文では、これらのパラメーターを明示的に指定し、イベントハンドラーのコンテキストをシミュレートできます。

備考

トリガー条件またはスケジュールには、CREATE EVENT 文でアクションを関連付けます。TRIGGER EVENT 文を使用すると、スケジュールされた時刻またはトリガー条件が発生していなくても、イベントハンドラーを強制的に実行することができます。TRIGGER EVENT は無効にされたイベントハンドラーを実行しません。

各 *value* は文字列です。各 *value* の最大長は、-gp サーバーオプションで指定された最大ページサイズによって制限されます。*value* の長さがページサイズを超える場合、ページが一杯になった時点で文字列はトランケートされます。

パーミッション

DBA 権限

関連する動作

なし

参照

- 「-gp dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「ALTER EVENT 文」417 ページ
- 「CREATE EVENT 文」540 ページ
- 「EVENT_PARAMETER 関数 [システム]」223 ページ

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、イベントに文字列パラメーターを渡す方法を示します。イベントは、トリガーされた時刻をデータベースサーバーメッセージウィンドウに表示します。

```
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' || event_parameter( 'time' );
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string( current timestamp ) );
```

TRUNCATE 文

テーブル定義を削除しないで、テーブルからすべてのローを削除します。

構文

TRUNCATE
TABLE [*owner*.]*table-name*
| **MATERIALIZED VIEW** [*owner*.]*materialized-view-name*

備考

TRUNCATE 文は、テーブルまたはマテリアライズドビューからすべてのローを削除します。

> **注意**
> TRUNCATE TABLE 文はテーブルからすべてのローを削除するため、同期またはレプリケーションに関連するデータベースでは慎重に使用してください。この文は DELETE 文と似ていますが、WHERE 句を指定できません。ただし、TRUNCATE 文の結果としてトリガーは発生しません。さらに、ローの削除はトランザクションログに入力されないため、同期またはレプリケーションが行われません。結果として、同期やレプリケーションが失敗するような矛盾が発生する可能性があります。

TRUNCATE 文を実行した後、オブジェクトのスキーマとすべてのインデックスは、DROP 文が実行されるまで引き続き存在します。スキーマ定義と制約はそのまま残り、トリガーとパーミッションは有効なままです。

table-name には、ベーステーブルまたはテンポラリテーブルの名前を指定できます。

TRUNCATE TABLE を使用すると、次の基準のすべてが満たされている場合に、高速方式のテーブルトランケーションが実行されます。

●そのテーブルへの外部キーも、そのテーブルからの外部キーも存在しない。

●TRUNCATE TABLE 文がトリガー内で実行されない。

●TRUNCATE TABLE 文がアトミック文の中で実行されない。

高速トランケーションが実行される場合、個々の DELETE はトランザクションログに記録されません。また、このオペレーションの前後に COMMIT が実行されます。スナップショットトランザクション内では高速トランケーションを使用できません。

即時テキストインデックスが構築されているテーブルか、または即時ビューから参照されているテーブルで TRUNCATE TABLE を使用しようとすると、トランケーションが失敗します。即時テキストインデックス以外のインデックスとマテリアライズドビューでは、このようなことは発生しません。ただし、従属するインデックスやマテリアライズドビューのデータをトランケートしてからテーブルで TRUNCATE TABLE 文を実行し、その後、インデックスとマテリアライズドビューをリフレッシュすることを強くおすすめします。

パーミッション

- テーブル所有者であるか、DBA 権限またはテーブルに対する ALTER パーミッションが必要です。

- ベーステーブルとマテリアライズドビューの場合、TRUNCATE 文は、操作がアトミックであるため (すべてのローが削除されるか、まったくされないか)、テーブルへの排他的なアクセスが必要です。つまり、トランケートされるテーブルを参照する、事前に開かれたカーソルを閉じ、COMMIT または ROLLBACK を実行して、テーブルへの参照を解除します。

- テンポラリテーブルの場合、各ユーザーはデータの独自のコピーを所有しているため、TRUNCATE 文を実行するときに排他的アクセスは不要です。

関連する動作

- マテリアライズドビューをトランケートする場合は、ビューのステータスを未初期化に変更します。

- TRUNCATE 文は、トリガー削除を実行しません。

- COMMIT は TRUNCATE 文の実行前後に実行されます。

- ローの個々の削除はトランザクションログには記録されません。したがって、TRUNCATE オペレーションはレプリケートされません。SQL Remote レプリケーションまたは Mobile Link リモートデータベースでは、この文を使用しないでください。

- テーブルに DEFAULT AUTOINCREMENT または DEFAULT GLOBAL AUTOINCREMENT と定義されたカラムがある場合、トランケーション操作はそのカラムの次に使用可能な値をリセットします。

参照

- 「DELETE 文」697 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「テーブルからのすべてのローの削除」『SQL Anywhere サーバー SQL の使用法』
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』
- 「高度：マテリアライズドビューのステータスとプロパティ」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008** TRUNCATE TABLE 文は、SQL/2008 標準のオプション言語機能 F200 です。TRUNCATE MATERIALIZED VIEW はベンダー拡張です。

例

Departments テーブルからすべてのローを削除します。

```
TRUNCATE TABLE Departments;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

TRUNCATE TEXT INDEX 文

MANUAL または AUTO REFRESH テキストインデックスのデータを削除します。

構文

TRUNCATE TEXT INDEX *text-index-name*
ON [*owner*.]*table-name*

パラメーター

ON 句　テキストインデックスを構築するテーブルの名前。

備考

TRUNCATE TEXT INDEX 文は、テキストインデックス定義を削除しないで、手動テキストインデックスからデータを削除する場合に使用します。たとえば、テキストインデックスのテキスト設定オブジェクトを変更してストップリストを更新する場合、最初にテキストインデックスをトランケートし、そのインデックスが参照しているテキスト設定オブジェクトを変更します。次にテキストインデックスをリフレッシュして、新しいデータをインデックスに移植する必要があります。

IMMEDIATE REFRESH (デフォルト) と定義されているテキストインデックスに対して TRUNCATE TEXT INDEX 文を実行することはできません。IMMEDIATE REFRESH テキストインデックスの場合は、インデックスを削除する必要があります。

パーミッション

- テキストインデックスが構築されているテーブルの所有者であるか、DBA 権限を持つか、またはテーブルへの ALTER パーミッションがあることが必要です。

- TRUNCATE TEXT INDEX ではテーブルへの排他アクセスが必要です。つまり、トランケートされるテーブルを参照するカーソルが開いている場合は、それらのカーソルをすべて閉じ、COMMIT または ROLLBACK 文を実行してテーブルへの参照を解除します。

関連する動作

オートコミット。

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキストインデックスの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE TEXT INDEX 文」667 ページ
- 「ALTER TEXT INDEX 文」480 ページ
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

最初の文は、txt_index_manual テキストインデックスを作成します。2 番目の文では、テキストインデックスにデータを移植します。3 番目の文では、テキストインデックスデータをトランケートします。

```
CREATE TEXT INDEX txt_index_manual ON MarketingInformation ( Description )
  MANUAL REFRESH;
REFRESH TEXT INDEX txt_index_manual ON MarketingInformation;
TRUNCATE TEXT INDEX txt_index_manual ON MarketingInformation;
```

トランケートされたテキストインデックスは、次回リフレッシュされるとき、データが再移植されます。

UNION 文

2 つ以上の SELECT 文またはクエリ式の結果を結合します。

構文

[**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*, ... **)**]

query-block : 「一般的な SQL 構文要素」403 ページを参照してください。

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value :
hostvar (許可されたインジケーター)
| *string*
| *identifier*
| *number*

パラメーター

FOR XML 句　FOR XML 句の詳細については、「SELECT 文」895 ページを参照してください。

OPTION 句　この句は、文を実行するためのヒントを指定するときに使用します。次のヒントがサポートされます。

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- *option-name* = *option-value* クエリテキスト内の OPTION(isolation_level = ...) の指定は、クエリの独立性レベルを指定する他のいずれの手段よりも優先されることに注意してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

UNION ALL は、2 つのクエリブロックの結果を連結して、単一の (より大きな) 結果セットにします。それぞれのクエリブロックはネストされることがあります。UNION DISTINCT は最終結果から重複ローを削除します。重複ローを削除するには余分な処理が必要なため、可能であれば UNION の代わりに UNION ALL を使用してください。UNION DISTINCT は UNION と同じです。

2 つの *query-block* の結果セットが UNION 対応になっている必要があります。つまり、それぞれの SELECT リストに同じ数の項目があり、個々の式のデータ型が比較可能となっている必要があります。2 つの SELECT リスト内の対応する項目が異なるデータ型の場合、SQL Anywhere は結果の中から対応するカラムのデータ型を選択し、各 *query-block* のカラムを自動的にそれぞれ変換します。

表示されるカラム名は最初の *query-block* に表示されるカラム名と同じであり、これらの名前は ORDER BY 句と一致する式の名前を判断するために使用されます。結果セットのカラム名をカスタマイズする別の方法として、共通のテーブル式 (WITH 句) を使用する方法があります。

パーミッション

各 *query-block* には SELECT パーミッションが必要です。

関連する動作

なし

参照

- 「EXCEPT 文」738 ページ
- 「INTERSECT 文」810 ページ
- OPTION 句、SELECT 文 902 ページ
- 「SELECT 文」895 ページ
- 「集合演算子と NULL」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　UNION は SQL/2008 標準のコア機能です。UNION とともに DISTINCT キーワードを明示的に指定する方法は、オプションの SQL 言語機能 T551 です。UNION とともに ORDER BY 句を指定する方法は、SQL 言語機能 F850 です。ORDER BY 句を含む *query-block* は、SQL/2008 機能 F851 の構成要素です。ロー制限句 (SELECT TOP または LIMIT) を含むクエリブロックは、コンテキストに応じて、オプションの SQL 言語機能 F857 または F858 を構成します。FOR XML 句と OPTION 句はベンダー拡張です。

- **Transact-SQL**　UNION と UNION ALL は Adaptive Server Enterprise によってサポートされます。FOR XML 句と OPTION 句はサポートされません。

例

従業員と顧客のそれぞれの姓すべてをリストします。

```
SELECT Surname
FROM Employees
UNION
SELECT Surname
FROM Customers;
```

UNLOAD 文

データソースからファイルにデータをアンロードします。

構文

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*

構文

unload-option :
APPEND { **ON** | **OFF** }
| **BYTE ORDER MARK** { **ON** | **OFF** }
| { **COMPRESSED** | **NOT COMPRESSED** }
| **COLUMN DELIMITED BY** *string*
| **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** }
| **ROW DELIMITED BY** *string*

encoding : *string*

パラメーター

TO 句　データのアンロード先ファイルの名前。*filename* パスは、データベースサーバーの起動ディレクトリへの相対パスです。このファイルが存在しない場合は、作成されます。ファイルがすでに存在する場合は、APPEND ON が指定されていないかぎり、そのファイルが上書きされます。

INTO FILE 句　セマンティック上は TO *filename* と同じです。

INTO CLIENT FILE 句　データのアンロード先となるクライアントコンピューター上のファイル。このファイルが存在しない場合は作成されます。ファイルがすでに存在する場合は、APPEND ON が指定されていないかぎり、そのファイルが上書きされます。パスは、クライアン

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

トコンピューター上で、クライアントアプリケーションの現在の作業フォルダーとの相対パスとして解決されます。

INTO VARIABLE 句　データのアンロード先変数。変数がすでに存在し、CHAR、NCHAR、または BINARY 型である必要があります。APPEND オプションを指定すると、アンロードされたデータが変数の現在の内容に連結されます。

APPEND 句　APPEND を ON にすると、アンロードされたデータは、指定されたファイルの最後に追加されます。APPEND を OFF にすると、指定されたファイルの内容がアンロードされたデータに置換されます。APPEND はデフォルトでは OFF になっています。COMPRESSED または ENCRYPTED 句を指定したときは、この句を指定できません。また、追加されるファイルが圧縮または暗号化されているときは使用できません。

BYTE ORDER MARK 句　この句は、バイトオーダーマーク (BOM) を書き込むどうかを制御するときに指定します。アンロードの送信先がローカルファイルまたはクライアントファイルの場合、このオプションはデフォルトで ON です。BYTE ORDER MARK オプションが ON で ENCODING が UTF-8 または UTF-16 の場合、BOM は書き込まれません。BYTE ORDER MARK が OFF の場合、BOM はアンロードされません。

COMPRESSED 句　データを圧縮するかどうかを指定します。デフォルトは NOT COMPRESSED です。データを追加する場合 (APPEND ON) は、データを圧縮できません。

COLUMN DELIMITED BY および DELIMITED BY 句　カラム間で使用される文字列です。デフォルトのカラムデリミターはカンマです。最長で 255 バイトの文字列を指定して、代替のカラムデリミターを指定できます。

ENCODING 句　すべてのデータベースデータは、データベースの文字エンコードから、指定した CHAR または NCHAR エンコードへと変換されます。ENCODING が指定されていない場合、データベースの CHAR エンコードが使用されます。

変換エラーがアンロード操作時に発生した場合、on_charset_conversion_failure オプションの設定に基づいてレポートされます。

次の例では、UTF-8 文字エンコードを使用してデータをアンロードします。

```
UNLOAD TABLE mytable TO 'mytable_data_in_utf8.dat' ENCODING 'UTF-8';
```

データにバイトオーダーマークを含めるには、BYTE ORDER 句を指定します。

ENCRYPTED 句　データを暗号化するかどうかを指定します。NOT ENCRYPTED (デフォルト) を指定した場合、データは暗号化されません。キーは指定するがアルゴリズムは指定しないで ENCRYPTED KEY を使用すると、データは AES128 と指定したキーを使用して暗号化されます。キーとアルゴリズムを指定して ENCRYPTED KEY を使用すると、データは指定したキーとアルゴリズムを使用して暗号化されます。CREATE DATABASE 文で許可されたアルゴリズムであれば、どのアルゴリズムでも使用できます。単純暗号化は指定できません。

データを追加する場合 (APPEND ON)、データは暗号化できません。

ESCAPES 句　ESCAPES を ON (デフォルト) にすると、データベースサーバーはエスケープ文字を書き込みます。改行文字は ¥n との組み合わせとして書き込まれ、他の文字はタブ文字の ¥x09 のような 16 進の ASCII のコードとしてデータにインクルードされます。2 つの円記号 (¥)

は 1 つの円記号として書き込まれます。円記号 (¥) の後に n、x、X、¥ 以外の文字がある場合、それらは別々の文字として書き込まれます。たとえば、¥q であれば、円記号と q が挿入されます。エスケープ文字に指定する文字列は 1 マルチバイト文字を超えないようにすることをおすすめします。

FORMAT 句　データを TEXT フォーマットまたは BCP アウトフォーマットで出力します。TEXT を選択すると、出力行はテキスト文字として書き込まれ、1 行あたり 1 つのローで構成され、カラムデリミター文字列によって値が区切られます。BCP を選択すると、BLOB などのデータが、Adaptive Server Enterprise で使用する BCP 入力ファイルとしてエクスポートされます。デフォルトのフォーマットは TEXT です。

HEXADECIMAL 句　デフォルトでは、HEXADECIMAL は ON です。バイナリカラム値は 0x*nnnnnn*... の形式で書き込まれます。ここで、0x は 1 つのゼロの後に 1 つの x が続きます。*n* は 16 進数の数字です。マルチバイト文字セットを扱う場合は、HEXADECIMAL ON を使用することが重要です。

HEXADECIMAL 句を使用する場合は、FORMAT TEXT 句のみ指定してください。

ESCAPE CHARACTER 句　この句は、データで使用するエスケープ文字を指定するときに使用します。16 進のコードと記号として書き込まれている文字に使用するデフォルトのエスケープ文字は円記号 (¥) です。たとえば、¥x0A は改行文字です。エスケープ文字は、ESCAPE CHARACTER 句を使って変更することができます。たとえば、感嘆符 (!) をエスケープ文字として使用するには、次のように入力します。

```
ESCAPE CHARACTER '!'
```

エスケープ文字に指定する文字列は 1 マルチバイト文字を超えないようにすることをお勧めします。

ORDER 句　ORDER を ON (デフォルト) に設定すると、エクスポートされたデータはクラスタードインデックス (存在する場合) の順に配列されます。クラスタードインデックスが存在しない場合、エクスポートされたデータはプライマリキー値で順序付けられます。ORDER を OFF に設定すると、ORDER BY 句を使わずにテーブルから選択したときと同じ順序でデータがエクスポートされます。ORDER を ON に設定すると、エクスポートは遅くなります。ただし、LOAD TABLE 文を使って再ロードする方が、インデックス手順が単純であるため速くなります。

UNLOAD *select-statement* の場合、ORDER 句は無視されます。ただし、SELECT 文で ORDER BY 句を指定すると、データを順序付けできます。

QUOTE 句　QUOTE 句は TEXT データ専用です。*string* は文字列値を囲みます。デフォルトは一重引用符 (アポストロフィ) です。

QUOTES 句　QUOTES をオン (デフォルト) に設定すると、エクスポートされる文字列がすべて引用符文字列 (デフォルトでは、一重引用符 (アポストロフィ)) で囲まれます。

ROW DELIMITED BY 句　この句は、レコードの末尾を示す文字列を指定するときに使用します。デフォルトのデリミター文字列は改行文字 (¥n) です。ただし、長さ 255 バイトまでの任意の文字列を使用できます。たとえば、... ROW DELIMITED BY '###' ... のように指定します。タブで区切った値を指定する場合、タブ文字を表す 16 進のエスケープシーケンス (9) を使用して、... ROW DELIMITED BY '¥x09' ... のように指定します。デリミター文字列に ¥n が含まれる場合、¥r¥n または ¥n のどちらかに一致します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

select-statement 文では、SELECT 文からのデータをカンマ区切りのファイルにエクスポートできます。SELECT 文に ORDER BY 句が指定されていない場合は、結果セットを順序付けできません。

UNLOAD TABLE 文は、データベーステーブルまたはマテリアライズドビューからファイルへの効率的な大量エクスポートができます。UNLOAD TABLE 文は、Interactive SQL 文の OUTPUT より効率的で、すべてのクライアントアプリケーションから呼び出すことができます。

INTO FILE または INTO CLIENT FILE が指定されたかどうかに応じて、データベースサーバーまたはクライアントアプリケーションはそれぞれ、指定されたファイルに対する書き込みのオペレーティングシステムパーミッションが付与されている必要があります。

バイナリデータ型のテーブルカラムをアンロードする場合、UNLOAD TABLE は **¥x***nnnn* の形式で 16 進文字列を書き込みます。*n* は 16 進数字です。

プロキシテーブルを持つデータベースをアンロードして再ロードする場合は、ユーザーがローカルデータベースとリモートデータベースの両方に対して同じパスワードを持っていても、外部ログインを作成して、ローカルユーザーをリモートユーザーにマッピングしてください。外部ログインがない場合は、リモートサーバーに接続できないため再ロードが失敗します。

変数をアンロードする場合 (INTO VARIABLE)、出力は次のように文字セットに変換されます。

- **CHAR**　変数に CHAR エンコードで書き込みます。ENCODING 句は CHAR エンコードと一致する必要があります。
- **NCHAR**　変数に NCHAR エンコードで書き込みます。ENCODING 句は NCHAR エンコードと一致する必要があります。
- **BINARY**　変数に BINARY エンコードで書き込みます。ENCODING は BINARY エンコーディングと一致する必要があります。それ以外の場合、CHAR エンコーディングが使用されます。

アンロードされるデータの圧縮と暗号化を選択すると、最初にデータが圧縮されます。

UNLOAD TABLE は、テーブル全体またはマテリアライズドビュー全体に排他ロックをかけます。

この文の実行中に、進行状況メッセージの表示を要求できます。

また、Progress 接続プロパティを使用して、文がどの程度実行されたかを確認することもできます。

日付値の最大精度を保持するには、date_format を YYYY-MM-DD に設定します。

TIMESTAMP 値の最大精度を保持するには、timestamp_format を YYYY-MM-DD HH:NN:SS.SSSSSS に設定します。

TIMESTAMP WITH TIMEZONE 値の最大精度を保持するには、timestamp_with_time_zone_format を YYYY-MM-DD HH:NN:SS.SSSSSS+HH:NN に設定します。

パーミッション

変数をアンロードする場合、(データソースへのアクセスに必要な通常のパーミッション以外の)パーミッションは必要ありません。

UNLOAD 文を実行するのに必要なパーミッションは、データベースサーバーの -gl オプションによって次のように異なります。

- -gl オプションが ALL に設定されている場合、UNLOAD 文内で参照されるテーブルの SELECT パーミッションが必要です。
- -gl オプションが DBA に設定されている場合、DBA 権限を持っていることが必要です。
- -gl オプションが NONE に設定されている場合、UNLOAD は使用できません。

クライアントコンピューターにあるファイルに書き込む場合

- WRITECLIENTFILE 権限が必要です。
- 書き込み先のディレクトリに書き込みパーミッションが必要です。
- allow_write_client_file データベースオプションが有効になっている必要があります。
- write_client_file セキュア機能が有効になっている必要があります。

関連する動作

なしこのクエリは現在の独立性レベルで実行されます。

参照

- 「CREATE DATABASE 文」521 ページ
- 「LOAD TABLE 文」814 ページ
- 「PASSTHROUGH 文 [SQL Remote]」855 ページ
- 「クラスタードインデックス」『SQL Anywhere サーバー SQL の使用法』
- 「サポートされている文字セット」『SQL Anywhere サーバー データベース管理』
- 「OUTPUT 文 [Interactive SQL]」848 ページ
- 「UNLOAD 文を使用したデータのエクスポート」『SQL Anywhere サーバー SQL の使用法』
- 「クライアントコンピューター上のデータへのアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「データのインポートとエクスポート」『SQL Anywhere サーバー SQL の使用法』
- 「UNLOAD TABLE 文を使用したデータのエクスポート」『SQL Anywhere サーバー SQL の使用法』
- 「-sf dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「allow_write_client_file オプション」『SQL Anywhere サーバー データベース管理』
- 「-gl dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「timestamp_format オプション」『SQL Anywhere サーバー データベース管理』
- 「date_format オプション」『SQL Anywhere サーバー データベース管理』
- 「on_charset_conversion_failure オプション」『SQL Anywhere サーバー データベース管理』
- Progress 接続プロパティ『SQL Anywhere サーバー データベース管理』
- 「progress_messages オプション」『SQL Anywhere サーバー データベース管理』
- 「WRITECLIENTFILE 権限」『SQL Anywhere サーバー データベース管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** ベンダー拡張。

例

次の例は、Products テーブルの内容を、UTF-8 でコード化されたファイル *productsT.dat* にアンロードします。

```
UNLOAD TABLE Products TO 'productsT.dat' ENCODING 'UTF-8';
```

次の例は、変数 @myProducts を作成してから、Products.Name カラムの変数をアンロードします。

```
CREATE VARIABLE @myProducts LONG VARCHAR;
UNLOAD SELECT NAME FROM Products INTO VARIABLE @myProducts;
```

UPDATE (位置付け) 文 [ESQL] [SP]

カーソルの現在位置のデータを変更します。

構文 1 [ESQL のみ]

UPDATE WHERE CURRENT OF *cursor-name*
{ **USING** [**SQL**] **DESCRIPTOR** *sqlda-name* | { [**FROM**] | [**USING**] } *hostvar-list* }

構文 2

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* (テーブルまたはビューの名前)

sqlda-name : *identifier*

パラメーター

USING DESCRIPTOR 句 変数に代入する場合、その変数がすでに宣言済みであることが必要です。また、変数名は「アット」記号 (@) で始めます。変数の代入とカラムの代入は混在させることができ、任意の数を使用できます。SET リストの代入の左辺にある名前が、更新されたテーブルのカラムと変数名に一致する場合、UPDATE 文はカラムを更新します。

SET 句 *set-item* で参照されるカラムは、更新対象となるテーブルまたはビューに配置してください。エイリアスや、他のテーブルまたはビューからのカラムは参照できません。更新するテーブルまたはビューにカーソル指定で相関名が与えられている場合は、SET 句に相関名を使用してください。

各 *set-item* は 1 つの *update-table* に関連付けられ、カーソルのクエリで一致するテーブルの対応するカラムは修正されます。*expression* は UPDATE リストに指定されているテーブルのカラムを参照します。また、+、-、...、COALESCE、IF などの演算子を使用すると、定数、ホスト変数、変数、クエリの SELECT リストからの式、またはそれらを組み合わせて使用できます。式は、カーソルのクエリから式のエイリアスを参照できません。また、UPDATE リストにあるカーソルのクエリの他のテーブルにあるカラムを参照することもできません。subselect、サブクエリ述部、集合関数は、*set-items* に使用できません。

各 *update-table* は、次のように、カーソルのクエリにあるテーブルとマッチングされます。

○相関名を指定した場合、同じ *table-or-view-name* と同じ *correlation-name* を持つカーソルのクエリにあるテーブルとマッチングされます。

○それ以外の場合、相関名が指定されていない同じ *table-or-view-name* を持つカーソルのクエリにテーブルがあるとき、または *table-or-view-name* と同じ相関名を持つとき、更新テーブルはカーソルのクエリのテーブルとマッチングされます。

○それ以外の場合、更新テーブルと同じ *table-or-view-name* を持つカーソルのクエリに単一のテーブルがある場合、更新テーブルはカーソルのクエリにあるテーブルとマッチングされます。

カラムにデフォルトが定義されている場合は、SET 句を使用してカラムをデフォルト値に設定できます。この例については、「UPDATE 文」976 ページの例の項を参照してください。

備考

この形式の UPDATE 文は、指定されたカーソルの現在のローを更新します。現在のローを、カーソルからフェッチされた最後のローとして定義します。カーソルに対する最後のオペレーションを位置付けられた DELETE 文にしないでください。

構文 1 では、SQLDA からのカラム、またはホスト変数リストからの値は、指定したカーソルから返されるカラムに 1 対 1 で対応します。SQLDA の sqldata ポインターが NULL ポインターの場合、該当する SELECT リスト項目は更新されません。

構文 2 では、要求されたカラムは指定されたクエリの現在のローに指定された値に設定されています。カラムは、指定した開いているカーソルの SELECT リストの中になくてもかまいません。このフォーマットは準備できます。

また、変数に代入する場合、その変数がすでに宣言済みであることが必要です。また、変数名は「アット」記号 (@) で始めます。変数の代入とカラムの代入は混在させることができ、任意の数を使用できます。SET リストの代入の左辺にある名前が、更新されたテーブルのカラムと変数名に一致する場合、UPDATE 文はカラムを更新します。

USING DESCRIPTOR、FROM *hostvar-list*、*hostvar* 形式を使用できるのは、ESQL の場合だけです。

パーミッション

更新するカラムに対する UPDATE パーミッションが必要です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

なし

参照

- 「INSERT 文」802 ページ
- 「LOAD TABLE 文」814 ページ
- 「MERGE 文」834 ページ
- 「DELETE 文」697 ページ
- 「DELETE 文 (位置付け) [ESQL] [SP]」695 ページ
- 「UPDATE 文」976 ページ

標準と互換性

- **SQL/2008**　構文 1 はベンダー拡張です。構文 2 は SQL/2008 標準のコア機能です。Embedded SQL プログラムで使用される構文 2 は、オプションの SQL 言語機能 B031、"Basic dynamic SQL" の一部です。更新されるテーブルを複数指定できる機能は、ベンダー拡張です。

  更新できるカーソルの範囲は、ansi_update_constraints オプションの設定によって異なります。順序付けされている (つまり、SQL クエリに ORDER BY 句が指定されている) カーソルの位置付け更新を実行する機能は、オプションの SQL/2008 言語機能 F831 "Full cursor update" を構成します。より複雑な SQL 構成での位置付け更新の実行には、追加のベンダー拡張が関与することがあります。

例

次は、UPDATE 文の WHERE CURRENT OF カーソルの例です。

```
UPDATE Employees
SET Surname = 'Jones'
WHERE CURRENT OF emp_cursor;
```

UPDATE 文 [SQL Remote]

データベース内のデータを変更します。

構文 1

UPDATE *table-list*
SET *column-name* **=** *expression*, ...
[**VERIFY (** *column-name*, ... **) VALUES (** *expression*, ... **)**]
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**], ...]

構文 2

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*

パラメーター

table-name *table-name* は、修正する必要がある、リモートデータベースのテーブルです。

publication-name *publication-name* は、サブスクリプションを変更する必要のあるパブリケーションです。

subscription-expression *subscription-expression* の値は、ローの新しい受信者と既存の受信者を決定するため、SQL Remote によって使用されます。*subscription-expression* は、値またはサブクエリのいずれかです。または、OLD と NEW の両方のサブスクリプション式を指定できます。

WHERE 句 WHERE 句は、サブスクリプションを作成したデータベース間で移動させるローを指定します。

備考

UPDATE 文を使って、1 つまたは複数のテーブルを修正します。それぞれ指定したカラムを、等号の右側の式の値に設定します。*expression* には制限がありません。*column-name* も式の中で使用できます。この場合、古い値を使用します。

WHERE 句を指定しない場合、すべてのローが更新されます。WHERE 句を指定する場合、探索条件を満たすローだけが更新されます。

通常、ローを更新する順序は問題ではありません。ただし、NUMBER(*) 関数と一緒に使って、ある指定された順序でローの中の数字を増加させる場合に、順序付けが役に立ちます。また、テーブルのプライマリキー値に 1 を追加するような操作を行う場合は、操作中にプライマリキーの重複が起こらないように、プライマリキーの降順でその操作を行う必要があります。

ビューを定義する SELECT 文の中に GROUP BY 句や集合関数がないか、または UNION 句を伴わない場合は、ビューを更新できます。

テーブルに挿入した文字列は、テーブルが大文字と小文字を区別するかどうかにかかわらず、常に入力された大文字と小文字がそのまま格納されます。そのため、文字列 Value で更新した文字データ型カラムは、常に大文字の V と残りの文字が小文字でデータベースに格納されます。SELECT 文は、文字列を Value として返します。ただし、データベースで大文字と小文字が区別されない場合は、すべての比較において Value は value、VALUE などと同じと見なされます。さらに、単一カラムのプライマリキーにエントリ Value がある場合は、プライマリキーがユニークでなくなるので、INSERT 文による value の追加は拒否されます。

オプションの FROM 句で、ジョインに基づいてテーブルを更新できます。FROM 句がある場合、WHERE 句は FROM 句のローが条件を満たしているかどうかを調べます。データの更新は、UPDATE キーワードのすぐ後のテーブルリストの中だけで行われます。

FROM 句を使用する場合、更新されるテーブル名をその文の両方の部分で同じように修飾することが重要です。一方の場所で相関名が使用されている場合は、もう一方の場所でも同じ相関名を使用します。修飾方法が異なると、エラーが発生します。

構文 1 と構文 2 は SQL Remote のみに適用されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 2 で OLD SUBSCRIBE BY 式と NEW SUBSCRIBE BY 式を付けない場合は BEFORE トリガーで使用する必要があります。

構文 2 で OLD SUBSCRIBE BY 式と NEW SUBSCRIBE BY 式を付ける場合はどこでも使用できます。

構文 1 は、SQL Remote だけのための構文であり、Message Agent によって 1 つのローの更新が実行されます。VERIFY 句には、更新するローの中にあると予想される値のセットを含めます。値が一致しない場合、UPDATE が処理される前に RESOLVE UPDATE トリガーが起動されます。VERIFY 句の照合が失敗しても、UPDATE は失敗しません。VERIFY 句を指定すると、テーブルが一度に 1 つずつ更新されます。

構文 2 は、必ず SQL Remote と一緒に使用します。構文 3 に OLD 式も NEW 式も使用しない場合は、関連する値にアクセスできるように、BEFORE トリガー内で使用します。この目的は、リストが変更されるたびに、SUBSCRIBE BY 値の完全なリストを提供することです。これは、SQL Remote トリガーに配置され、データベースサーバーが SUBSCRIBE BY 値の現在のリストを計算できるようにします。両方のリストがトランザクションログ内に記録されます。

Message Agent は 2 つのリストを使用して、ローが必要な、ローを保有しない任意のリモートデータベースにローが移動したことを確認します。また、Message Agent は、ローを保有し、そのローが不要になったリモートデータベースからローを削除します。そのローを保有し、依然そのローを必要とするリモートデータベースは、UPDATE 文の影響を受けません。

UPDATE 文の構文 2 を使用すると、古い SUBSCRIBE BY リストと新しい SUBSCRIBE BY リストを明示的に指定して、SQL Remote トリガーの効率を高めることができます。このリストがないと、データベースサーバーはパブリケーション定義から古い SUBSCRIBE BY リストを計算します。通常は、新しい SUBSCRIBE BY リストは古い SUBSCRIBE BY リストとほんのわずか異なるだけなので、古いリストの処理は 2 回行われます。古いリストと新しいリストの両方を指定すると、この無駄な作業を避けることができます。

OLD SUBSCRIBE BY 構文と NEW SUBSCRIBE BY 構文は、同じ SUBSCRIBE BY 式で同じトリガーを使用し、たくさんのテーブルを更新する場合には特に便利です。これにより、パフォーマンスが大幅に向上します。

SUBSCRIBE BY 式は、値またはサブクエリのいずれかです。

UPDATE 文の構文 2 は、特定の SQL Remote 機能を実装するために使用したり、BEFORE トリガー内で使用したりします。

SUBSCRIBE BY 句内のサブクエリを使って作成したパブリケーションについては、UPDATE 文の構文 2 を含むトリガーを作成してローがそれぞれの正しいサブスクリプションに格納されていることを確認します。

UPDATE 文の構文 2 を使うと、トランザクションログ内にエントリが作成されますが、データベーステーブルは変更されません。

パーミッション

修正対象のカラムに対する UPDATE パーミッションが必要です。

関連する動作

なし

参照

- 「BEFORE UPDATE トリガー」『SQL Remote』

標準と互換性

- **SQL/2008** ベンダー拡張。

例

従業員 Philip Chin (従業員 129) を販売部からマーケティング部に異動する例を示します。

```
UPDATE Employees
VERIFY( DepartmentID ) VALUES( 300 )
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

UPDATE 文

データベーステーブルにある既存のローを変更します。

構文 1

UPDATE [*row-limitation*] *table-expression* [, ...]
SET *set-item*[, ...]
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**] , ...]
[**OPTION(** *query-hint*, ... **)**]

table-expression :ジョイン、外部ジョイン、ビュー、派生テーブルを含むことのできるテーブル式。「FROM 句」758 ページを参照してください。

構文 2

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* [, ...] **)**]

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文 3

UPDATE *table-name*
SET *set-item*, ...
VERIFY (*column-name*, ... **) VALUES (** *expression*, ... **)**
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**], ...]
[**OPTION(** *query-hint*, ... **)**]

構文 4

UPDATE [*owner.*]*table-name*
PUBLICATION *publication*
{ **SUBSCRIBE BY** *expression*
| **OLD SUBSCRIBE BY** *expression* **NEW SUBSCRIBE BY** *expression*
}
WHERE *search-condition*

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*)

set-item :
[*correlation-name.*]*column-name* **=** { *expression* | **DEFAULT** }
| [*owner-name.*]*table-name.column-name* **=** { *expression* | **DEFAULT** }
| **@***variable-name* **=** *expression*

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| **FORCE NO OPTIMIZATION**
| *option-name* = *option-value*

table-name :
[*owner.*]*base-table-name*
| *temporary-table-name*
| *derived-table-name*
| [*owner.*]*view-name*

option-name : *identifier*

option-value :
hostvar (許可されたインジケーター)
| *string*
| *identifier*
| *number*

パラメーター

UPDATE 句　構文 1 と構文 2 では、*table-expression* にテンポラリテーブル、グローバルテンポラリテーブル、派生テーブル、またはビューを含めることができます。ビューと派生テーブルは、更新可能でないものを除き、更新することができます。構文 1 では、複数の *table-expression* のリストが、基本となるテーブル式でフォーマットされたローの直積になります。これは、WHERE 句を使用して制限することができます。構文 1 と構文 2 はどちらもジョインの更新を許可します。構文 3 および構文 4 では、*table-name* はベーステーブルであることが必要です。

ビューを定義しているクエリ指定が更新可能な場合にかぎり、ビューに対して更新操作を実行できます。

***row-limitation* 句**　ローを制限する句を使用することによって、アップデートされるローを WHERE 句を満たすローのサブセットのみに制限できます。TOP 引数と START AT 引数には、ホスト変数、整数定数、または整数変数を使用した簡単な算術演算を指定できます。TOP 引数は 0 以上である必要があります。START AT 引数は 0 より大きい値にする必要があります。ローの順序を意味のあるものにするために ORDER BY 句も指定します。

SET 句　SET 句は、カラムと値の変更方法を指定します。

SET 句は、このフォーマットを使用するカラムを計算カラム値に設定するときに使用できます。

```
SET column-name = expression, ...
```

それぞれ指定したカラムを、等号の右側の式の値に設定します。*expression* には制限がありません。式が *column-name* である場合は、そのカラムの古い値が使われます。

カラムにデフォルトが定義されている場合は、SET 句を使用してカラムをデフォルト値に設定できます。この使用例については、例の項を参照してください。

また、SET 句で次のフォーマットを使用して変数に代入することもできます。

```
SET @variable-name = expression, ...
```

変数に値を代入する場合、その変数がすでに宣言済みであることが必要です。また、変数名はアット記号 (@) で始めます。変数名が更新されるテーブルのカラム名と一致している場合、UPDATE 文はカラム値を更新し、変数は変更しないでそのままにします。変数の代入とカラムの代入は任意の順序で混在させることができます。

FROM 句　FROM *table-expression* 句を使用すると、ジョインに基づいてテーブルを更新できます。*table-expression* には、OUTER、CROSS、NATURAL ジョインなどの任意の複雑なテーブル式を指定できます。

FROM 句が存在する場合、*table-name* では更新される単一のテーブルを指定します。この名前は FROM 句と同じように修飾されている必要があります。FROM 句で相関名が使用されている場合、同一の相関名を *table-name* として指定します。更新されるテーブル式が派生テーブルの場合、派生テーブルを *table-name* の指定で繰り返す必要があります。

ansi_update_constraints オプションが Strict に設定されている場合、構文 2 は使用できません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

FROM 句が指定されている場合、SET 句では更新される *table-name* のカラムのみを指定できます。修飾方法が異なると、エラーが発生します。

次の文は、構文 2 を使用した UPDATE 文に相関名を使用したテーブル式が含まれており、この文のテーブル名に潜在的なあいまいさが存在することを示しています。

```
UPDATE table_1
SET column_1 = ...
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

ここで、FROM 句の table_1 の各インスタンスには相関名があり、table_1 のそれ自体へのセルフジョインを示しています。ただし、この UPDATE 文はセルフジョインを構成するローのうち、どのローを更新対象とするかの指定に失敗します。これは、次のように、UPDATE 文の相関名を指定することで修正できます。

```
UPDATE table_1 as alias_1
SET column_1 = ...
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

WHERE 句　WHERE 句を指定すると、探索条件を満たすローだけが更新されます。WHERE 句を指定しない場合、すべてのローが更新されます。

ORDER BY 句　通常、ローを更新する順序は重要ではありません。ただし、FIRST 句または TOP 句と一緒に使用する場合は、ローを更新する順序が意味を持ちます。

ORDER BY 句ではカラムの順序数を使用できません。

ORDER BY 句を使用するときは、ansi_update_constraints オプションを Strict に設定しないでください。

ORDER BY 句に含まれるカラムを更新するには、ansi_update_constraints オプションを Off に設定します。

OPTION 句　この句は、文を実行するためのヒントを指定するときに使用します。次のヒントがサポートされます。

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- *option-name* = *option-value* クエリテキスト内の OPTION(isolation_level = ...) の指定は、クエリの独立性レベルを指定する他のいずれの手段よりも優先されることに注意してください。

備考

テーブルに挿入した文字列は、データベースが大文字と小文字を区別するかどうかに関係なく、常に入力された大文字と小文字の状態のままで格納されます。文字列 Street で更新した CHAR データ型カラムは、常に大文字の S と残りの文字が小文字でデータベースに格納されます。SELECT 文は、文字列を Street として返します。ただし、データベースで大文字と小文字が区別されない場合は、すべての比較において Street は street、STREET などと同じと見なされます。さらに、単一カラムのプライマリキーにエントリ Street がある場合、プライマリキーがユニークでなくなるため、別のローのプライマリキーの street への UPDATE は拒否されます。

新しい値が元の値と同じ場合、データは変更されません。ただし、BEFORE UPDATE トリガーは、ローを対象に UPDATE が実行されるたび、新しい値が古い値と異なるかどうかに関係なく起動されます。一方、AFTER UPDATE トリガーは、新しい値が古い値と異なる場合にのみ起動されます。

構文 3 と構文 4 は SQL Remote だけに適用できます。

構文 3 は、SQL Remote だけのための構文であり、Message Agent によって 1 つのテーブルの 1 つのローの更新が実行されます。VERIFY 句には、更新するローの中にあると予想される値のセットを含めます。値が一致しない場合、UPDATE が処理される前に RESOLVE UPDATE トリガーが起動されます。UPDATE は失敗しませんが、それは単に VERIFY 句が一致に失敗するからです。

UPDATE 文の構文 4 は、特定の SQL Remote 機能を実装するために使用したり、BEFORE トリガー内で使用したりします。この文は、リストが変更されるたびに、SUBSCRIBE BY 値の完全なリストを提供します。これは、SQL Remote トリガーに配置され、データベースサーバーが SUBSCRIBE BY 値の現在のリストを計算できるようにします。両方のリストがトランザクションログ内に記録されます。

Message Agent は 2 つのリストを使用して、ローが必要な、ローを保有しない任意のリモートデータベースにローが移動したことを確認します。また、Message Agent は、ローを保有し、そのローが不要になったリモートデータベースからローを削除します。そのローを保有し、依然そのローを必要とするリモートデータベースは、UPDATE 文の影響を受けません。

SUBSCRIBE BY 句内のサブクエリを使って作成したパブリケーションについては、UPDATE 文の構文 4 を含むトリガーを作成してローがそれぞれの正しいサブスクリプションに格納されていることを確認します。

UPDATE 文の構文 4 を使用すると、古い SUBSCRIBE BY リストと新しい SUBSCRIBE BY リストを明示的に指定して、SQL Remote トリガーの効率を高めることができます。このリストがないと、データベースサーバーはパブリケーション定義から古い SUBSCRIBE BY リストを計算します。通常は、新しい SUBSCRIBE BY リストは古い SUBSCRIBE BY リストとほんのわずか異なるだけなので、古いリストの処理は 2 回行われます。古いものと新しいリストの両方を指定すると、この余分な作業を避けることができます。

SUBSCRIBE BY 式は、値またはサブクエリのいずれかです。

UPDATE 文の構文 4 を使うと、トランザクションログ内にエントリが作成されますが、データベーステーブルは変更されません。

UPDATE 文を使用してデータを大量に更新する場合も、カラム統計は更新されます。

パーミッション

修正対象のカラムに対する UPDATE パーミッションが必要です。

関連する動作

カラム統計は変更された値を反映するように更新されます。

テーブルにプライマリキー、UNIQUE 制約、または UNIQUE インデックスがある場合、一意性制約に違反しなければテーブル変更を実行できないときには、UPDATE 文の処理にテンポラリ

テーブルを使用する必要がある場合があります。テンポラリテーブルには、1 つ以上の一意性制約に違反している UPDATE 文で変更されたローが格納されます。これらのローは、UPDATE 文の実行時に一時的にベーステーブルから削除され、後で再挿入されます。この動作は、AFTER トリガーと他の同時接続に影響することがあります。

参照

- 「DELETE 文」697 ページ
- 「DELETE 文 (位置付け) [ESQL] [SP]」695 ページ
- 「INSERT 文」802 ページ
- 「UPDATE 文 [SQL Remote]」973 ページ
- 「CREATE TRIGGER 文」670 ページ
- 「CREATE TABLE 文」651 ページ
- 「PUT 文 [ESQL]」860 ページ
- OPTION 句、SELECT 文 902 ページ
- 「MERGE 文」834 ページ
- 「ALTER TABLE 文」462 ページ
- 「UPDATE (位置付け) 文 [ESQL] [SP]」971 ページ
- 「VERIFY 句を持つ UPDATE 文」『SQL Remote』
- 「TSEQUAL 関数 [システム] (旧式)」377 ページ
- 「FROM 句」758 ページ
- 「ビュー」『SQL Anywhere サーバー SQL の使用法』
- 「ジョイン：複数テーブルからのデータ検索」『SQL Anywhere サーバー SQL の使用法』
- 「更新時のロック」『SQL Anywhere サーバー SQL の使用法』
- 「UPDATE 文のレプリケーション」『SQL Remote』
- 「SELECT、UPDATE、DELETE クエリブロック内のロー制限句」『SQL Anywhere サーバー SQL の使用法』
- 「UPDATE によるデータの変更」『SQL Anywhere サーバー SQL の使用法』
- 「ansi_update_constraints オプション」『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　UPDATE 文の構文 1 は SQL/2008 標準のコア機能です。構文 2 はベンダー拡張です。構文 3 と 4 は、SQL Remote だけに対するベンダー拡張です。

  構文 1 には、次の 2 つのオプションの SQL 言語機能のサポートが含まれます。

  ○1 つ以上の派生テーブルを含む可能性のあるジョインの更新に対するサポートは、オプションの SQL 言語機能 T111、"Updatable joins, unions, and columns" の一部です。

  ○UPDATE 文の検索条件部分を形成するネストされたサブクエリで参照されるテーブルの変更に対するサポートは、オプションの SQL/2008 言語機能 F781 "Self-referencing operations" を構成します。

  構文 1 の次の機能はベンダー拡張です。

  ○FROM 句と ORDER BY 句。

  ○*row-limitation* 句。

  ○複数の *table-expression* を指定する機能。

○SET 句を使用して変数を更新する機能。

○OPTION 句。

構文 1 では、ansi_update_constraints オプションの設定によって、変更できるテーブル式の形式が制御されます。SQL/2008 コア機能との互換性を確保するために、ansi_update_constraints オプションは必ず Strict に設定してください。

例

次の例では、サンプルデータベースを使用して、従業員 Philip Chin (employeeID = 129) を販売部からマーケティング部に異動する例を示します。

```
UPDATE Employees
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

次の例では、サンプルデータベースを使用して、既存するすべての受注の ID から 2000 を引いて番号をふり直します。

```
UPDATE SalesOrders AS orders
SET orders.ID = orders.ID - 2000
ORDER BY orders.ID ASC;
```

この更新が可能となるのは、SalesOrderItems テーブル (プライマリキー SalesOrders.ID を参照する) の外部キーがアクション ON UPDATE CASCADE を使用して定義されている場合だけです。SalesOrderItems テーブルも更新されます。文で ORDER BY 句が指定されており、SET 句でも順序属性が指定されているため、ansi_update_constraints オプションを Off に設定しないとエラーが返されます。

3 つめの例は、サンプルデータベースを使用して、データベースの現在の独立性レベル設定ではなく、独立性レベル 2 で製品の価格を変更します。

```
UPDATE Products
SET UnitPrice = 7.00
WHERE ID = 501
OPTION( isolation_level = 2 );
```

4 つめの例では再度サンプルデータベースを使用し、構文 2 を使用して、現在の在庫を上回る量の注文が 1 つ以上存在する Tee Shirt の在庫をリセットします。

```
UPDATE Products AS a
SET Quantity = 0
FROM Products a JOIN SalesOrderItems b ON a.ID = b.ProductID
WHERE a.Name = 'Tee Shirt' AND b.Quantity > a.Quantity;
```

5 つめの例では、構文 1 を使用して、これらの Tee Shirts の在庫をリセットし、Tee Shirt 注文の ShipDate を今日の日付にリセットします。

```
UPDATE Products a JOIN 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
```

4 つめと 5 つめの例では、ansi_update_constraints オプションを Strict 以外の値に設定する必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

6 つめの例では、テーブルを更新して、カラムをデフォルト値に設定する方法を示します。この例では、テーブル MyTable を作成し、このテーブルにデータを移植します。次に、SET 句を指定して UPDATE 文を実行し、カラム値をデフォルトに変更します。

```
CREATE 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 @@dbts );

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;

UPDATE MyTable SET  LastTime = DEFAULT, LastTimestamp = DEFAULT
  WHERE TableName LIKE '%sys%';
```

VALIDATE 文

現在のデータベース、または現在のデータベース内にある単一のテーブルやマテリアライズドビューまたはインデックスを検証します。

構文 1 - データベースの検証

VALIDATE { **CHECKSUM** | **DATABASE** }

構文 2 - テーブルとマテリアライズドビューの検証

VALIDATE {
TABLE [*owner*.]*table-name*
| **MATERIALIZED VIEW** [*owner*.]*materialized-view-name* }
[**WITH EXPRESS CHECK**]

構文 3 - インデックスの検証

VALIDATE {
INDEX *index-name*
| [**INDEX**] **FOREIGN KEY** *role-name*
| [**INDEX**] **PRIMARY KEY** }
ON [*owner*.]*object-name*
}

object-name :
table-name
| *materialized-view-name*

構文 4 - テキストインデックスの検証

VALIDATE TEXT INDEX *index-name*
ON [*owner*.]*table-name*

備考

VALIDATE CHECKSUM 文は、データベース上で各ページのチェックサムの検証を実行するときに使用します。VALIDATE CHECKSUM 文によって、データベースページがディスク上で変更されていないことを確認します。チェックサムを有効にしてデータベースを作成すると、各データベースページがディスクに書き込まれる前に、そのページのチェックサムが計算されます。VALIDATE CHECKSUM は、各データベースページをデータベースサーバーのキャッシュからではなく、ディスクから読み込み、各ページのチェックサムを計算します。ページに対して計算されたチェックサムが、そのページについて格納されているチェックサムと一致しない場合は、エラーが発生し、無効なページに関する情報がデータベースサーバーメッセージウィンドウに表示されます。VALIDATE CHECKSUM 文は、チェックサムが無効になっているデータベースにはお勧めしません。

VALIDATE DATABASE 文を使用して、フリーマップでページが割り当て済みかフリーかが正しく識別され、孤立している BLOB がないことを確認します。VALIDATE DATABASE は、チェックサム検証を実行して、各データベースページが正しいオブジェクトに所属することを検証します。たとえば、テーブルページ上でテーブル ID によって、定義がテーブルページのセットに最新のテーブルを含む有効なテーブルが識別されなければなりません。VALIDATE DATABASE 文はページをサーバーのキャッシュに順番に転送します。このため、検証時に、データベースサーバーはキャッシュに転送されたページの内容とチェックサムを常に検証します。データベースクリーナーが実行している間にデータベースの検証を開始すると、データベースクリーナーの実行が終了するまで検証は実行されません。

VALIDATE TABLE 文は、ベーステーブル内の一連のローと値が各インデックスに含まれる一連のローと値に一致するかどうかを確認して、指定されたテーブルとそのすべてのインデックスを検証します。また、すべてのテーブルの BLOB を移動して、BLOB のアロケーションマップを検証し、孤立した BLOB を検出します。また、VALIDATE TABLE 文は、すべてのテーブルのインデックスページの物理構造を確認し、インデックスハッシュ値の順序、インデックスの一意性要件も (指定されている場合は) 検査します。外部キーインデックスの場合、WITH EXPRESS CHECK 句が指定されていなければ、プライマリキーテーブルの各値が検索されて、参照整合性制約がそのまま残っていることが確認されます。VALIDATE TABLE 文は、VALIDATE DATABASE と同じようにデータベースサーバーのキャッシュを使用するため、データベースサーバーもテーブルとそのインデックスで使用されるすべてのページのチェックサムと基本的な妥当性を検査します。

VALIDATE INDEX 文は VALIDATE TABLE 文と同じ操作を実行しますが、指定されたインデックスのみ検証して、基になるテーブルやその他のインデックスは検証しない点が異なります。外部キーインデックスの場合、WITH EXPRESS CHECK 句が指定されていなければ、プライマリキーテーブルの各値が検索されて、参照整合性制約がそのまま残っていることが確認されます。WITH EXPRESS CHECK 句を指定すると、参照整合性検査が無効になるため、パフォーマンスが大幅に向上する場合があります。指定されたインデックスが外部キーインデックスでない場合、WITH EXPRESS CHECK は効果がありません。

VALIDATE TEXT INDEX 文を使用して、インデックス内の単語の位置情報が正常であることを確認できます。位置情報が維持されていない場合は、エラーが生成され、テキストインデックスを再作成する必要があります。テキストインデックスが自動または手動のいずれかの場合、REFRESH TEXT INDEX 文を実行して、テキストインデックスを再作成できます。

生成されたエラーが即時テキストインデックスに関わる場合、即時インデックスを削除して、新しいインデックスを作成する必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **警告**
> テーブルまたはデータベース全体の検証は、データベースに変更を加えている接続がない場合に実行してください。そうしないと、実際に破損していなくても、何らかの形でデータベースが破損したことを示すエラーがレポートされます。

パーミッション

DBA 権限または VALIDATE 権限

関連する動作

なし

参照

- 「REFRESH TEXT INDEX 文」870 ページ
- 「DROP TEXT INDEX 文」733 ページ
- 「CREATE TEXT INDEX 文」667 ページ
- 「検証ユーティリティ (dbvalid)」『SQL Anywhere サーバー データベース管理』
- 「sa_validate システムプロシージャー」1210 ページ
- 「sa_clean_database システムプロシージャー」1046 ページ
- 「データベースの検証」『SQL Anywhere サーバー データベース管理』
- 「CREATE DATABASE 文」521 ページ
- 「CREATE INDEX 文」567 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

WAITFOR 文

指定された時間の間、または指定の時間になるまで現在の接続処理を遅らせます。

構文

WAITFOR {
DELAY *time*
| **TIME** *time* }
[**CHECK EVERY** *integer*]
[**AFTER MESSAGE BREAK**]

time : *string*

パラメーター

DELAY 句　DELAY を使用すると、処理は特定の期間だけ中断されます。

TIME 句　TIME を指定すると、データベースサーバー時刻が指定の時刻に達するまで、処理が中断されます。現在のサーバー時刻が指定の時刻より後の場合は、翌日のその時刻になるまで処理が中断されます。

CHECK EVERY 句　このオプション句は、WAITFOR 文が起動する頻度を制御します。デフォルトでは、5 秒ごとに起動します。値はミリ秒単位であり、最小値は 250 ミリ秒です。

AFTER MESSAGE BREAK 句　WAITFOR 文を使用して、別の接続からのメッセージを待つことができます。メッセージが受信されると、通常このメッセージは WAITFOR 文を実行したアプリケーションに転送され、WAITFOR 文は待機を継続します。AFTER MESSAGE BREAK 句が指定されている場合、別の接続からのメッセージが受信されると、WAITFOR 文が完了します。メッセージテキストはアプリケーションに転送されませんが、MessageReceived 接続プロパティの値を取得してアクセスできます。

備考

WAITFOR 文は、定期的 (デフォルトでは 5 秒おき) に起動して、キャンセルされたかどうか、またはメッセージが受信されたかどうかをチェックします。どちらも発生していない場合、文は待機を継続します。

スケジュールされたイベントはそれ専用の接続で実行されるため、通常は WAITFOR TIME よりスケジュールされたイベントを使用することをおすすめします。

パーミッション

なし

関連する動作

WAITFOR 文の実装によって、文を処理しているワーカーは、文が待機している間ブロックされます。これによって、ワーカープール内の使用可能なワーカーの数が減少します。

参照

- 「SQL Anywhere のスレッド化」『SQL Anywhere サーバー データベース管理』
- 「CREATE EVENT 文」540 ページ
- 「MESSAGE 文」841 ページ
- MessageReceived 接続プロパティ『SQL Anywhere サーバー データベース管理』

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の例は 3 秒だけ待機します。

```
WAITFOR DELAY '00:00:03';
```

次の例は 0.5 秒 (500 ミリ秒) だけ待機します。

```
WAITFOR DELAY '00:00:00:500';
```

次の例は午後 8 時まで待機します。

```
WAITFOR TIME '20:00';
```

次の例では、接続 1 の WAITFOR 文は、接続 2 からメッセージを受信すると完了します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
// 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
```

WHENEVER 文 [ESQL]

Embedded SQL プログラム内でのエラー処理を指定します。

構文

WHENEVER {
SQLERROR
| **SQLWARNING**
| **NOTFOUND** }
GOTO
  label
  | **STOP**
  | **CONTINUE**
  | { *C-code*; }

label : *identifier*

備考

WHENEVER 文を使って、SQL 文を処理するときにデータベースに発生するエラー、警告、例外条件をトラップします。Embedded SQL C プログラム内のどこにでもこの文を入れることができ、この文はコードを生成しません。プリプロセッサーは、それぞれの継続する SQL 文の後にコードを生成します。すべての Embedded SQL 文では、WHENEVER 文のソース行から同じエラー条件の次の WHENEVER 文、またはソースファイルの最後まで、エラーアクションは有効です。

> **注意**
> エラー条件は、文がいつ実行されたかではなく、C 言語ソースファイル内での位置に基づいて有効になります。

デフォルトアクションは CONTINUE です。

この文は、単純なプログラムの中で便利に使用できます。たいていの場合、SQLCA (SQLCODE) の sqlcode フィールドを直接チェックするのがエラー条件をチェックする最も簡単な方法です。この場合、WHENEVER 文を使用しません。WHENEVER 文が実行されると、プリプロセッサーはそれぞれの文の後に *if (SQLCODE)* テストを生成します。

パーミッション

なし

関連する動作

なし

標準と互換性

- **SQL/2008**　WHENEVER 文で実行される例外条件宣言は、SQL/2008 標準のコア機能です。標準では、キーワード SQLERROR ではなく SQLEXCEPTION を使用します。WHENEVER 文に単なる文ラベルではなく直接 C コードを含める機能は、ベンダー拡張です。STOP アクションもベンダー拡張です。

例

次は、WHENEVER 文の例です。

```
EXEC SQL WHENEVER NOTFOUND GOTO done;
EXEC SQL WHENEVER SQLERROR
  {
    PrintError( &sqlca );
    return( FALSE );
  };
```

WHILE 文 [T-SQL]

文または複合文を繰り返し実行します。

構文

WHILE *search-condition statement*

備考

WHILE は、単一の SQL 文とキーワード BEGIN と END で囲まれた複合文を制御します。

複合文中での文の実行は、BREAK 文と CONTINUE 文で制御できます。BREAK 文はループを終了し、ループの最後を示す END の後から実行が再開されます。CONTINUE 文は、その後の文をすべて省略して WHILE ループを再開します。

パーミッション

なし

関連する動作

なし

参照

- 「LOOP 文」832 ページ
- 「 CONTINUE 文」520 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

標準と互換性

- **SQL/2008** Transact-SQL 拡張。WHILE 文は、オプションの SQL/2008 言語機能 P002 "Computational completeness" の一部です。WHILE 文の Transact-SQL 変形には、END WHILE は含まれていません。

例

次のコードは、WHILE の使い方を示します。

```
WHILE ( SELECT AVG(UnitPrice) FROM Products ) < $30
BEGIN
  UPDATE Products
  SET UnitPrice = UnitPrice + 2
  IF ( SELECT MAX(UnitPrice) FROM Products ) > $50
    BREAK
END
```

BREAK 文は、最も高い製品の価格が 50 ドルを超える場合、WHILE ループをブレークします。そうでない場合、ループは平均価格が 30 ドルになるまで継続します。

WINDOW 句

SELECT 文の AVG や RANK などの Window 関数を使用するウィンドウのすべてまたは一部を定義します。

構文

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**

パラメーター

PARTITION BY 句 PARTITION BY 句は、指定した式のユニークな値に基づいて、結果セットを論理グループに構築します。Window 関数を使用してこの句を使用する場合、関数は独立して

各パーティションに適用されます。たとえば、カラム名が指定された PARTITION BY に従う場合、結果セットはカラムの個別の値によってパーティション化されます。

この句を省略すると、全体の結果セットはパーティションとして扱われます。

PARTITION BY *expression* には、整数リテラルを使用できません。

ORDER BY 句　ORDER BY 句は、結果セットの各パーティションのローをソートする方法を定義します。さらに、昇順の場合は ASC (デフォルト)、降順の場合は DESC を指定して順序を制御することができます。

ORDER BY *expression* には、整数リテラルを使用できません。

この句が省略されると、SQL Anywhere は最も効率的な順序でローを返します。そのため、ローに最後にアクセスした日時によって、結果セットでの表示順序が異なることがあります。

ROWS 句と RANGE 句　ROWS 句または RANGE 句を使用して、ウィンドウのサイズを表します。ウィンドウサイズは、パーティションの 1 つのロー、複数のロー、またはすべてのローに設定できます。ウィンドウサイズは、現在のローの値からのオフセットとしてデータ値の範囲を指定するか (RANGE)、現在のローからの物理的なロー数としてオフセットを指定する (ROWS) ことで表すことができます。

RANGE 句を使用する場合は、範囲計算で値の格納が必要になるため、ORDER BY も指定する必要があります。範囲のための ORDER BY 句には式が 1 つ含まれていること、およびその式の結果が日付または数値になることが必要です。

ROWS 句または RANGE 句を指定しなかった場合、データベースサーバーは、ORDER BY 句が指定されているかどうかに基づいて、デフォルトのウィンドウサイズを使用します。

○ **PRECEDING 句**　PRECEDING 句は、現在のローを参照ポイントとして使用して、ウィンドウの最初のローを定義するときに使用します。開始ローは、現在のローの前にあるロー数で表します。たとえば、5 PRECEDING と設定すると、ウィンドウは現在のローの前にある 5 番目のローから開始されます。

  UNBOUNDED PRECEDING は、ウィンドウの最初のローを設定し、パーティションの最初のローにするときに使用します。

○ **BETWEEN 句**　BETWEEN 句は、現在のローを参照ポイントとして使用して、ウィンドウの最初のローと最後のローを定義するときに使用します。最初のローと最後のローは、それぞれ現在のローの前にあるロー数と後にあるロー数で表します。たとえば、BETWEEN 3 PRECEDING AND 5 FOLLOWING と設定すると、ウィンドウは現在のローの前にある 3 番目のローから開始され、現在のローの後の 5 番目のローで終了します。

  BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING は、ウィンドウの最初のローと最後のローを設定することで、それぞれパーティションの最初のローと最後のローにするときに使用します。これは ROW 句や RANGE 句を指定する場合のデフォルト動作と同じです。

○ **FOLLOWING 句**　FOLLOWING 句は、現在のローを参照ポイントとして使用して、ウィンドウの最後のローを定義するときに使用します。最後のローは、現在のローの後にあるロー数で表します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

UNBOUNDED FOLLOWING は、ウィンドウの最後のローを設定し、パーティションの最後のローにするときに使用します。

備考

WINDOW 句は、SELECT 文の ORDER BY 句の前に指定する必要があります。

LIST 関数には例外がありますが、すべての集合関数を Window 関数として使用できます。ただし、ランキング集合関数 (RANK、DENSE_RANK、PERCENT_RANK、CUME_DIST、ROW_NUMBER) には ORDER BY 句が必要です。また、WINDOW 句やインライン定義には ROW 句と RANGE 句は使用できません。それ以外の Window 関数では、任意の句を使用できます。

参照

- 「SELECT 文」895 ページ
- 「OLAP のサポート」『SQL Anywhere サーバー SQL の使用法』
- 「ウィンドウ定義」『SQL Anywhere サーバー SQL の使用法』
- 「ウィンドウ定義 : OVER 句と WINDOW 句を使用したインライン定義」『SQL Anywhere サーバー SQL の使用法』

標準と互換性

- **SQL/2008**　WINDOW 句と Window 集合関数は、SQL/2008 のオプション言語機能 T611 "Elementary OLAP operations" と T612 "Advanced OLAP operations" を構成します。Window 関数 FIRST_VALUE と LAST_VALUE は、ベンダー拡張です。

例

次の例は、選択した州に在住する全従業員について、各従業員の給料と平均給料を返します。結果は State、Surname の順にソートされます。

```
SELECT EmployeeID, Surname, Salary, State,
  AVG( Salary ) OVER Salary_Window
FROM Employees
WINDOW Salary_Window AS ( PARTITION BY State )
ORDER BY State, Surname;
```

WRITETEXT 文 [T-SQL]

CHAR、NCHAR、または BINARY カラムのログなしの更新を許可します。この機能は Transact-SQL との互換性のためにのみ提供されているものであり、使用しないことをおすすめします。

構文

WRITETEXT *table-name.column-name*
text-pointer [**WITH LOG**] *data*

備考

既存のカラム値を更新します。更新は、WITH LOG オプションが提供されないかぎり、トランザクションログに記録されません。ビューに対して WRITETEXT 操作を実行することはできません。

パーミッション

なし

関連する動作

WRITETEXT はトリガーを起動しません。また、デフォルトでは、作業内容はトランザクションログに記録されません。

参照

- 「READTEXT 文 [T-SQL]」866 ページ
- 「TEXTPTR 関数 [テキストとイメージ]」368 ページ

標準と互換性

- **SQL/2008** Transact-SQL 拡張。

例

次の Embedded SQL コードフラグメントは、WRITETEXT 文の使い方を示しています。この例での SELECT 文は、単一のローを返します。次の例は、指定されたローの Description カラムの内容を新しい値で置き換えます。

```
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';
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

テーブル

システムテーブル

すべてのデータベースの構造は、複数のシステムテーブルに記述されています。システムテーブルは、ユーザー SYS によって所有されます。これらのテーブルの内容は、データベースサーバーでのみ変更できます。したがって、テーブルの内容の変更に UPDATE、DELETE、INSERT 文は使用できません。また、ALTER TABLE 文と DROP 文を使って、これらのテーブルの構造を変更することもできません。

システムテーブル内のデータは、対応する「システムビュー」経由で公開されます。システムテーブル名はそれぞれ大文字の i (I) で始まります。対応するそれぞれのシステムビューも同じ名前になりますが、最初の i は削除されます。

参照

- 「システムビュー」1253 ページ

DUMMY システムテーブル

カラム名	カラム型	カラム制約	テーブル制約
dummy_col	INTEGER	NOT NULL	

DUMMY テーブルは、常に 1 つだけのローを持つ、読み込み専用のテーブルとして提供されています。これはデータベースから情報を抽出するのに役立ちます。次に、データベースから現在のユーザー ID と今日の日付を取り出す例を示します。

```
SELECT USER, today(*) FROM SYS.DUMMY;

SELECT USER, today(*);
```

dummy_col　このカラムは使用されません。テーブルはカラムなしでは作成できないので、このカラムが存在します。

SYS.DUMMY テーブルからの読み取りコストは、同様のユーザー作成テーブルからの読み取りコストよりも低いです。これは、SYS.DUMMY のテーブルページにラッチがないためです。

アクセスプランは、SYS.DUMMY テーブルのスキャンによって構築されるわけではありません。そうではなく、SYS.DUMMY への参照がローコンストラクターアルゴリズムに置き換えられ、これがテーブル参照を仮想化します。これにより、SYS.DUMMY の使用に伴う競合を排除できます。ただし、DUMMY は、テーブル名か相関名またはその両方として、短いプラン、長いプラン、グラフィカルなプランに引き続き表示されます。

ISYSARTICLE システムテーブル

ISYSARTICLE システムテーブルの各ローは、パブリケーション内のアーティクルを示します。

参照

- 「SYSARTICLE システムビュー」1254 ページ

ISYSARTICLECOL システムテーブル

ISYSARTICLECOL システムテーブルの各ローは、アーティクル内のカラムを識別します。

参照

- 「SYSARTICLECOL システムビュー」1255 ページ

ISYSATTRIBUTE システムテーブル

このテーブルは内部でのみ使用されます。

ISYSATTRIBUTENAME システムテーブル

このテーブルは内部でのみ使用されます。

ISYSCAPABILITY システムテーブル

ISYSCAPABILITY システムテーブルの各ローは、リモートサーバーの機能を識別します。

参照

- 「SYSCAPABILITY システムビュー」1255 ページ

ISYSCHECK システムテーブル

ISYSCHECK システムテーブルの各ローは、テーブル内の名前付き検査制約を識別します。

参照

- 「SYSCHECK システムビュー」1256 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ISYSCOLPERM システムテーブル

ISYSCOLPERM システムテーブルの各ローは、カラムの UPDATE、SELECT、または REFERENCES パーミッションを示します。

参照

- 「SYSCOLPERM システムビュー」1257 ページ

ISYSCOLSTAT システムテーブル

ISYSCOLSTAT システムテーブルには、オプティマイザーが使用するカラムの統計情報が含まれます。

> **注意**
> SQL Anywhere 12 以降を使用して作成されたデータベースでは、不正なアクセスからデータを保護するため、このテーブルは常に暗号化されます。

参照

- 「SYSCOLSTAT システムビュー」1258 ページ

ISYSCONSTRAINT システムテーブル

ISYSCONSTRAINT システムテーブルの各ローは、システムテーブル以外のすべてのテーブル内の名前付き制約を示します。

参照

- 「SYSCONSTRAINT システムビュー」1259 ページ

ISYSDEPENDENCY システムテーブル

ISYSDEPENDENCY システムテーブルの各ローは、テーブルとビューの依存性を示します。

参照

- 「SYSDEPENDENCY システムビュー」1262 ページ

ISYSDBFILE システムテーブル

ISYSDBFILE システムテーブルの各ローは、DB 領域を示します。

参照

- 「SYSDBFILE システムビュー」1259 ページ

ISYSDBSPACE システムテーブル

ISYSDBSPACE システムテーブルの各ローは、DB 領域を示します。

参照

- 「SYSDBSPACE システムビュー」1260 ページ

ISYSDBSPACEPERM システムテーブル

ISYSDBSPACEPERM システムテーブルの各ローは、DB 領域のパーミッションを示します。

参照

- 「SYSDBSPACEPERM システムビュー」1261 ページ

ISYSDOMAIN システムテーブル

事前に定義されたデータ型 (「ドメイン」とも呼ばれます) には、ユニークな番号が割り当てられています。ISYSDOMAIN テーブルは参考用に用意され、これらの番号と対応するデータ型間の関係を示します。このテーブルは変更できません。

参照

- 「SYSDOMAIN システムビュー」1262 ページ

ISYSEVENT システムテーブル

ISYSEVENT システムテーブルの各ローは、CREATE EVENT で作成されたイベントを示します。

参照

- 「SYSEVENT システムビュー」1263 ページ

ISYSEXTERNLOGIN システムテーブル

ISYSEXTERNLOGIN システムテーブルの各ローは、リモートデータアクセスの外部ログインを示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **注意**
> SQL Anywhere 12 以降を使用して作成されたデータベースでは、不正なアクセスからデータを保護するため、このテーブルは常に暗号化されます。

参照

- 「SYSEXTERNLOGIN システムビュー」1267 ページ

ISYSFKEY システムテーブル

ISYSFKEY システムテーブルの各ローは、データベース内の外部キーを示します。

参照

- 「SYSFKEY システムビュー」1268 ページ

ISYSGROUP システムテーブル

ISYSGROUP システムテーブルの各ローは、グループのメンバーを定義します。このテーブルは、グループとメンバーの多対多の関係を示します。

参照

- 「SYSGROUP システムビュー」1269 ページ

ISYSHISTORY システムテーブル

ISYSHISTORY システムテーブルの各ローは、異なるバージョンのソフトウェアか異なるプラットフォーム、またはその両方でデータベースが開始された時間を示します。

参照

- 「SYSHISTORY システムビュー」1269 ページ

ISYSIDX システムテーブル

ISYSIDX システムテーブルの各ローは、データベース内のインデックスを示します。

参照

- 「SYSIDX システムビュー」1273 ページ

ISYSIDXCOL システムテーブル

ISYSIDXCOL システムテーブルの各ローは、インデックスのカラムを示します。

参照

- 「SYSIDXCOL システムビュー」1275 ページ

ISYSJAR システムテーブル

ISYSJAR システムテーブルの各ローは、システムの JAR ファイルを定義します。

参照

- 「SYSJAR システムビュー」1276 ページ

ISYSJARCOMPONENT システムテーブル

ISYSJARCOMPONENT システムテーブルの各ローは、JAR ファイルコンポーネントを定義します。

参照

- 「SYSJARCOMPONENT システムビュー」1277 ページ

ISYSJAVACLASS システムテーブル

ISYSJAVACLASS システムテーブルの各ローは、Java クラスを示します。

参照

- 「SYSJAVACLASS システムビュー」1277 ページ

ISYSLOGINMAP システムテーブル

ISYSLOGINMAP システムテーブルには、統合化ログインまたは Kerberos ログインを使用したデータベースへの接続に使用できるすべてのユーザープロファイル名が含まれます。セキュリティ上の理由から、このテーブルの内容は DBA 権限を持つユーザーだけが表示できます。

参照

- 「SYSLOGINMAP システムビュー」1278 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ISYSLOGINPOLICY システムテーブル

ISYSLOGINPOLICY システムテーブルの各ローは、ログインポリシーを示します。

参照

- 「SYSLOGINPOLICY システムビュー」1279 ページ

ISYSLOGINPOLICYOPTION システムテーブル

ISYSLOGINPOLICYOPTION システムテーブルの各ローは、ログインポリシーのオプションを示します。

参照

- 「SYSLOGINPOLICYOPTION システムビュー」1280 ページ

ISYSMIRROROPTION システムテーブル

ISYSMIRROROPTION システムテーブルの各ローは、ミラーサーバーのオプションに関する記述です。

参照

- 「SYSMIRROROPTION システムビュー」1280 ページ

ISYSMIRRORSERVER システムテーブル

ISYSMIRRORSERVER システムテーブルの各ローは、ミラーサーバーのオプションに関する記述です。

参照

- 「SYSMIRRORSERVER システムビュー」1280 ページ

ISYSMIRRORSERVEROPTION システムテーブル

SYSMIRRORSERVEROPTION システムテーブルの各ローは、ミラーサーバーのオプションに関する記述です。

参照

- 「SYSMIRRORSERVEROPTION システムビュー」1281 ページ

ISYSMVOPTION システムテーブル

ISYSMVOPTION システムテーブルの各ローには、データベースのマテリアライズドビューまたはテキストインデックスの作成オプションの値があります。

参照

- 「SYSMVOPTION システムビュー」1282 ページ

ISYSMVOPTIONNAME システムテーブル

ISYSMVOPTIONNAME システムテーブルの各ローには、ISYSMVOPTION にリストされている、マテリアライズドビューまたはテキストインデックスの作成オプションの名前があります。

参照

- 「SYSMVOPTIONNAME システムビュー」1283 ページ

ISYSOBJECT システムテーブル

ISYSOBJECT システムビューの各ローは、データベース内のオブジェクトを示します。データベースオブジェクトの例として、テーブル、ビュー、カラム、インデックス、プロシージャーなどがあります。

参照

- 「SYSOBJECT システムビュー」1283 ページ

ISYSOPTION システムテーブル

ISYSOPTION システムテーブルの各ローは、1 つのユーザー ID のオプション設定を示します。オプション設定は SET 文で ISYSOPTION テーブルに保存され、各ユーザーはオプションごとに独自の設定を指定できます。

参照

- 「SYSOPTION システムビュー」1285 ページ

ISYSOPTSTAT システムテーブル

ISYSOPTSTAT システムテーブルは、コストモデルの調整情報を、ALTER DATABASE CALIBRATE 文で計算して格納します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SYSOPTSTAT システムビュー」1285 ページ

ISYSPHYSIDX システムテーブル

ISYSPHYSIDX システムテーブルの各ローは、データベース内の物理インデックス 1 つを示します。

参照

- 「SYSPHYSIDX システムビュー」1285 ページ

ISYSPROCEDURE システムテーブル

ISYSPROCEDURE システムテーブルの各ローは、データベース内のプロシージャーを示します。

参照

- 「SYSPROCEDURE システムビュー」1286 ページ

ISYSPROCPARM システムテーブル

ISYSPROCPARM システムテーブルの各ローは、データベース内のプロシージャーに対するパラメーターを示します。

参照

- 「SYSPROCPARM システムビュー」1288 ページ

ISYSPROCPERM システムテーブル

ISYSPROCPERM テーブルの各ローは、あるプロシージャーを呼び出すための、ユーザーに付与されたパーミッションを示します。

参照

- 「SYSPROCPERM システムビュー」1290 ページ

ISYSPROXYTAB システムテーブル

ISYSPROXYTAB システムテーブルの各ローは、プロキシテーブルを示します。

参照

- 「SYSPROXYTAB システムビュー」1291 ページ

ISYSPUBLICATION システムテーブル

ISYSPUBLICATION システムテーブルの各ローは、SQL Remote または Mobile Link パブリケーションを示します。

参照

- 「SYSPUBLICATION システムビュー」1291 ページ

ISYSREMARK システムテーブル

ISYSREMARK システムテーブルの各ローは、オブジェクトの注釈 (コメント) を示します。

参照

- 「SYSREMARK システムビュー」1293 ページ

ISYSREMOTEOPTION システムテーブル

ISYSREMOTEOPTION システムテーブルの各ローは、SQL Remote メッセージリンクパラメーターの値を示します。

参照

- 「SYSREMOTEOPTION システムビュー」1293 ページ

ISYSREMOTEOPTIONTYPE システムテーブル

ISYSREMOTEOPTIONTYPE システムテーブルの各ローは、SQL Remote メッセージリンクパラメーター 1 つを示します。

参照

- 「SYSREMOTEOPTIONTYPE システムビュー」1294 ページ

ISYSREMOTETYPE システムテーブル

ISYSREMOTETYPE システムテーブルには、SQL Remote に関する情報があります。

参照

- 「SYSREMOTETYPE システムビュー」1294 ページ

ISYSREMOTEUSER システムテーブル

ISYSREMOTEUSER システムテーブルの各ローは、REMOTE パーミッションを持つユーザー ID (サブスクライバー) を示します。そのユーザーに送信された SQL Remote メッセージと、そのユーザーから送信されたメッセージのステータスも合わせて示します。

参照

- 「SYSREMOTEUSER システムビュー」1295 ページ
- 「REMOTE RESET 文 [SQL Remote]」874 ページ

ISYSSCHEDULE システムテーブル

ISYSSCHEDULE システムテーブルの各ローは、CREATE EVENT 文の SCHEDULE 句に指定されたイベントの起動時刻を示します。

参照

- 「SYSSCHEDULE システムビュー」1297 ページ

ISYSSEQUENCE システムテーブル

ISYSSEQUENCE システムテーブルには、各ユーザー定義シーケンスごとに 1 つのローがあります。

参照

- 「SYSSEQUENCE システムビュー」1299 ページ

ISYSSEQUENCEPERM システムテーブル

ISYSSEQUENCEPERM システムテーブルには、ユーザーまたはグループがシーケンスに対して保持する権限が記録されます。

参照

- 「SYSSEQUENCEPERM システムビュー」1300 ページ

ISYSSERVER システムテーブル

ISYSSERVER システムテーブルの各ローは、リモートサーバーを示します。

参照

- 「SYSSERVER システムビュー」1301 ページ

ISYSSOURCE システムテーブル

ISYSSOURCE システムテーブルの各ローには、ISYSOBJECT システムテーブルにリストされているオブジェクトのソースが含まれます。

参照

- 「SYSSOURCE システムビュー」1301 ページ

ISYSSPATIALREFERENCESYSTEM システムテーブル

ISYSSPATIALREFERENCESYSTEM システムテーブルの各ローは、データベースに定義されている空間参照系に関する記述です。

参照

- 「SYSSPATIALREFERENCESYSTEM システムビュー」1302 ページ

ISYSSQLSERVERTYPE システムテーブル

ISYSSQLSERVERTYPE システムテーブルには、Adaptive Server Enterprise との互換性に関する情報が含まれます。

参照

- 「SYSSQLSERVERTYPE システムビュー」1306 ページ

ISYSSUBSCRIPTION システムテーブル

ISYSSUBSCRIPTION システムテーブルの各ローは、REMOTE パーミッションを持つあるユーザー ID からの、あるパブリケーションに対するサブスクリプションを示します。

参照

- 「SYSSUBSCRIPTION システムビュー」1307 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ISYSSYNC システムテーブル

このテーブルには、Mobile Link 同期に関する情報が入っています。このテーブルの一部のカラムには、機密データが含まれている可能性があります。このため、このテーブルにアクセスできるのは DBA 権限を持つユーザーに限られます。SYSSYNC2 ビューを使用すると、機密データを含む可能性のあるカラムを除いて、このテーブルのデータにパブリックアクセスできます。

参照

- 「SYSSYNC システムビュー」1308 ページ

ISYSSYNCPROFILE システムテーブル

このテーブルには、Mobile Link 用の同期プロファイルに関する情報が入っています。

参照

- 「SYSSYNCPROFILE システムビュー」1309 ページ

ISYSSYNCSCRIPT システムテーブル

このテーブルには、Mobile Link 同期スクリプトに関する情報が入っています。

参照

- 「SYSSYNCSCRIPT システムビュー」1310 ページ

ISYSTAB システムテーブル

ISYSTAB システムテーブルの各ローは、データベース内のテーブル 1 つを示します。

参照

- 「SYSTAB システムビュー」1311 ページ

ISYSTABCOL システムテーブル

ISYSTABCOL システムテーブルの各ローは、データベース内のテーブルのカラムを示します。

参照

- 「SYSTABCOL システムビュー」1315 ページ

ISYSTEXTCONFIG システムテーブル

ISYSTEXTCONFIG システムテーブルの各ローは、全文検索機能で使用するテキスト設定を示します。

参照

- 「SYSTEXTCONFIG システムビュー」1320 ページ

ISYSTEXTIDX システムテーブル

ISYSTEXTIDX システムテーブルの各ローは、全文検索機能で使用するテキストインデックスを示します。

参照

- 「SYSTEXTIDX システムビュー」1322 ページ

ISYSTEXTIDXTAB システムテーブル

ISYSTEXTIDXTAB システムテーブルの各ローは、全文検索機能で使用するテキストインデックスを示します。

参照

- 「SYSTEXTIDXTAB システムビュー」1324 ページ

ISYSTABLEPERM システムテーブル

ISYSTABLEPERM システムテーブルの各ローは 1 つのテーブル、パーミッションを与えるユーザー ID (「grantor」)、そしてパーミッションを与えられるユーザー ID (「grantee」) に対応します。

参照

- 「SYSTABLEPERM システムビュー」1318 ページ

ISYSTRIGGER システムテーブル

ISYSTRIGGER システムテーブルの各ローは、データベース内のトリガーを示します。

参照

- 「SYSTRIGGER システムビュー」1324 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ISYSTYPEMAP システムテーブル

ISYSTYPEMAP システムテーブルには、ISYSSQLSERVERTYPE システムテーブルの互換性マッピング値があります。

参照

- 「SYSTYPEMAP システムビュー」1327 ページ

ISYSUNITOFMEASURE システムテーブル

ISYSUNITOFMEASURE システムテーブルの各ローは、データベースに定義されている測定単位に関する記述です。

参照

- 「SYSUNITOFMEASURE システムビュー」1327 ページ

ISYSUSER システムテーブル

ISYSUSER システムテーブルの各ローは、システム内のユーザーを示します。

> **注意**
> SQL Anywhere 12 以降では、不正なアクセスからデータを保護するため、このテーブルは常に暗号化されます。

参照

- 「SYSUSER システムビュー」1328 ページ

ISYSUSERAUTHORITY システムテーブル

ISYSUSERAUTHORITY システムテーブルの各ローは、ユーザーに付与された権限を示します。

参照

- 「SYSUSERAUTHORITY システムビュー」1329 ページ

ISYSUSERMESSAGE システムテーブル

ISYSUSERMESSAGE システムテーブルの各ローには、エラーに対するユーザー定義メッセージが格納されます。

参照

- 「SYSUSERMESSAGE システムビュー」1329 ページ

ISYSUSERTYPE システムテーブル

ISYSUSERTYPE システムテーブルの各ローは、ユーザー定義のデータ型を示します。

参照

- 「SYSUSERTYPE システムビュー」1330 ページ

ISYSVIEW システムテーブル

ISYSVIEW システムテーブルの各ローは、データベース内のビューを示します。

参照

- 「SYSVIEW システムビュー」1331 ページ

ISYSWEBSERVICE システムテーブル

ISYSWEBSERVICE システムテーブルの各ローは、Web サービスを示します。

参照

- 「SYSWEBSERVICE システムビュー」1333 ページ

診断トレーシングテーブル

次は、アプリケーションプロファイルと診断トレーシングに使用されるメインテーブルです。このテーブルは、dbo ユーザーが所有します。これらテーブルの多くには、似た名前と似たスキーマを持つグローバルで共有のテンポラリテーブルが存在します。たとえば、sa_diagnostic_blocking テーブルには、対応するグローバルなテンポラリテーブルとして sa_tmp_diagnostic_blocking table があり、スキーマは同じです。トレーシングセッション中に、診断データはこれらのテンポラリテーブルに書き込まれます。テンポラリテーブルはロギングされないため、トレーシングセッション中のパフォーマンスは優れています。サーバーに与える影響を最小限に抑えるには重要です。

参照

- 「アプリケーションプロファイリング」『SQL Anywhere サーバー SQL の使用法』
- 「診断トレーシング」『SQL Anywhere サーバー SQL の使用法』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

sa_diagnostic_auxiliary_catalog テーブル

sa_diagnostic_auxiliary_catalog テーブルは、dbo ユーザーが所有し、運用データベースとトレーシングデータベース間のデータベースオブジェクトのマッピングに使用されます。オブジェクトには、テーブル、プロシージャー、関数などがあります。このテーブルは、主にインデックスコンサルタントや TRACED_PLAN 関数に使用されます。

カラム

カラム名	カラム型	説明
original_object_id	UNSIGNED BIGINT	メイントレーシングデータベースにある、このオブジェクトのオブジェクト ID。
local_object_id	UNSIGNED BIGINT	補助トレーシングデータベースにある、このオブジェクトのオブジェクト ID。
pages_if_table	UNSIGNED INT	オブジェクトがテーブルの場合、これはテーブルのページ数です。オブジェクトがテーブル以外の場合、この値は NULL です。
rows_if_table	UNSIGNED BIGINT	オブジェクトがテーブルの場合、これはテーブルのロー数です。オブジェクトがテーブル以外の場合、この値は NULL です。

参照

- 「TRACED_PLAN 関数 [その他]」373 ページ
- 「インデックスコンサルタント」『SQL Anywhere サーバー SQL の使用法』

sa_diagnostic_blocking テーブル

sa_diagnostic_blocking テーブルは、dbo ユーザーが所有し、ブロッキングイベントを記録します。ブロッキングイベントのロギングが有効な場合、リソースにアクセスを試みている間、接続がブロックされるたびにローがこのテーブルに挿入されます。一般的に、この問題はテーブルまたはローのロックによって発生します。大量にブロックがある場合、テーブルとローの競合を軽減するために、アプリケーションの同時実行性を確認する必要があります。

このテーブルには sa_diagnostic_blocking と sa_tmp_diagnostic_blocking という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
lock_id	UNSIGNED BIGINT	ローまたはテーブルのロックがブロックを引き起こした場合、ブロックを引き起こしたロックの ID。それ以外の場合は NULL。
request_id	UNSIGNED BIGINT	ブロックがカーソルのために発生しなかった場合、ブロックされた要求の ID。それ以外の場合は NULL。この値は、sa_diagnostic_request の要求に割り当てられている ID に対応します。
cursor_id	UNSIGNED BIGINT	ブロックがカーソルのために派生した場合はカーソルの ID。それ以外の場合は NULL。この値は、sa_diagnostic_cursor のカーソルに割り当てられている ID に対応します。
original_table_object_id	UNSIGNED BIGINT	テーブルのロックのためにブロックが発生した場合、ブロックが発生したテーブルの ID。それ以外の場合は NULL。
rowid	UNSIGNED BIGINT	テーブルのロックのためにブロックが発生した場合、ブロックが発生したテーブルの ID。それ以外の場合は NULL。
block_time	TIMESTAMP	ブロックが発生した時間。
unblock_time	TIMESTAMP	ブロックが終了した時間。
blocked_by	UNSIGNED INT	ロックを保持し、ブロックを引き起こした接続の ID。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「トランザクションのブロックとデッドロック」『SQL Anywhere サーバー SQL の使用法』
- 「ロックの仕組み」『SQL Anywhere サーバー SQL の使用法』

sa_diagnostic_cachecontents テーブル

sa_diagnostic_cachecontents テーブルは、dbo ユーザーが所有します。診断トレーシングが有効なときは、キャッシュコンテンツのスナップショットが定期的に撮られます。
sa_diagnostic_cachecontents テーブルは、スナップショットを撮ったときのキャッシュ内の各テーブルのテーブルページ数、および各テーブルのロー数を記録します。オプティマイザーはこの情報を使用して、クエリで最初に最適化した条件を再作成し、最適化の決定を下します。

sa_diagnostic_cachecontents テーブルのデータは、クエリアクティビティがあれば、20 秒ごとに更新されます。

このテーブルには sa_diagnostic_cachecontents と sa_tmp_diagnostic_cachecontents という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
"time"	TIMESTAMP	キャッシュのスナップショットを撮った時間。
original_table_object_id	UNSIGNED BIGINT	スナップショットで表される各テーブルのオブジェクト ID。
pages_in_cache	UNSIGNED INT	スナップショットに指定したテーブルの場合、スナップショット時のキャッシュの総ページ数。
num_table_pages	UNSIGNED INT	スナップショットで指定したテーブルの場合、テーブルの総ページ数。
num_table_rows	UNSIGNED BIGINT	スナップショットで指定したテーブルの場合、テーブルの総ロー数。

sa_diagnostic_connection テーブル

sa_diagnostic_connection テーブルは、dbo ユーザーが所有しています。また、ロギングセッション中にアクティブに各データベース接続に 1 つのローがあります。接続と接続解除がロギングセッション中に発生した場合、その発生時間は sa_diagnostic_request テーブルから派生します。

このテーブルのほとんどの値は、接続プロパティの値を反映しています。

このテーブルには sa_diagnostic_connection と sa_tmp_diagnostic_connection という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
connection_number	UNSIGNED INT	データベースへの特定のユーザーの接続を識別するためにデータベースサーバーが割り当てる番号。この値は、Number 接続プロパティの値を反映しています。
connection_name	LONG VARCHAR	オプションの接続名プロパティ。この値は、Name 接続プロパティの値を反映しています。
user_name	LONG VARCHAR	データベースに接続するユーザー名。
comm_link	CHAR(40)	クライアント側のネットワークプロトコルオプションを指定します。この値は、CommLinks 接続プロパティの値を反映しています。
node_address	LONG VARCHAR	クライアント／サーバー接続のクライアント用ノード。この値は、NodeAddress 接続プロパティの値を反映しています。
appinfo	LONG VARCHAR	クライアントコンピューターの IP アドレス、稼働しているオペレーティングシステムなど、クライアント処理に関する情報。この値は、AppInfo 接続プロパティの値を反映しています。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』

sa_diagnostic_cursor テーブル

sa_diagnostic_cursor テーブルは、dbo ユーザーが所有します。各ローは、ロギングセッション中に開かれた内部カーソルまたは外部カーソルを示します。

このテーブルには sa_diagnostic_cursor と sa_tmp_diagnostic_cursor という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
cursor_id	UNSIGNED BIGINT	カーソルを識別するユニークな番号。
query_id	UNSIGNED BIGINT	このカーソルの範囲にあるクエリを識別します。
isolation_level	TINYINT	このカーソルを開いた独立性レベル。
flags	UNSIGNED INT	内部的に使用。
forward_fetches	UNSIGNED INT	カーソルで実行された、プリフェッチなどの転送フェッチの数。
reverse_fetches	UNSIGNED INT	カーソルで実行された、プリフェッチなどのリバースフェッチの数。
absolute_fetches	UNSIGNED INT	カーソルで実行された絶対フェッチの数。
first_fetch_time_ms	UNSIGNED INT	最初のローをフェッチするときにかかる時間。

カラム名	カラム型	説明
total_fetch_time_ms	UNSIGNED INT	フェッチにかかる時間。この値には、実際のフェッチ間 (シンクタイム) のアプリケーション処理時間を含みません。
plan_xml	LONG VARCHAR	カーソルを閉じたときにダンプされるカーソルの詳細プラン。このプランには、必要に応じて詳細な統計情報が含まれます。

参照

- 「カーソルの使用法」『SQL Anywhere サーバー プログラミング』

sa_diagnostic_deadlock テーブル

sa_diagnostic_deadlock テーブルは、dbo ユーザーが所有します。診断トレーシングが有効で、デッドロックイベントのトレーシングを含めるように設定されている場合、デッドロックが発生するたびに、ローセットがこのテーブルに挿入されます (デッドロックの一部である各接続の 1 ローが挿入されます。)。1 のデッドロックイベントを構成するすべてのローセットは、snapshot_id でユニークに識別されます。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
snapshot_id	UNSIGNED BIGINT	このローが含まれるデッドロックイベントを識別する番号。このカラムは、スナップショットアイソレーションと関係がないことに注意してください。
snapshot_at	TIMESTAMP	デッドロックが発生した時刻。
waiter	UNSIGNED INT	このローが表す接続の接続数。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	カラム型	説明
request_id	UNSIGNED BIGINT	デッドロックが発生したときにこの接続が処理していた要求 ID。
original_table_object_id	UNSIGNED BIGINT	この接続がブロックされたときのテーブルのオブジェクト ID。
rowid	UNSIGNED BIGINT	この接続がブロックされたときのローのレコード ID。
owner	UNSIGNED INT	このローをロックした接続の接続数。
rollback_operation_count	UNSIGNED INT	コミットされていないオペレーションの数。

参照

- 「トランザクションのブロックとデッドロック」『SQL Anywhere サーバー SQL の使用法』

sa_diagnostic_hostvariable テーブル

sa_diagnostic_hostvariable テーブルは dbo ユーザーが所有しています。また、指定したカーソルが使用するホスト変数値が含まれます。

このテーブルには sa_diagnostic_hostvariable と sa_tmp_diagnostic_hostvariable という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
request_id	UNSIGNED BIGINT	ホスト変数が所属する要求の ID。
cursor_id	UNSIGNED BIGINT	ホスト変数が保持するカーソルの ID。
hostvar_num	UNSIGNED SMALLINT	SQL 文のホスト変数の順序位置。

カラム名	カラム型	説明
hostvar_type	UNSIGNED TINYINT	ホスト変数のドメイン数。通常、文字列、整数、または浮動。
hostvar_value	LONG NVARCHAR	ホスト変数の値を表す文字列。ホスト変数が整数または浮動の場合でも、この値は文字列として表されます。

参照

- 「Embedded SQL のホスト変数」『SQL Anywhere サーバー プログラミング』

sa_diagnostic_internalvariable テーブル

sa_diagnostic_internalvariable テーブルは dbo ユーザーが所有しています。また、指定した文が使用する内部 (ローカル) 変数値が含まれます。このテーブルは、主にインデックスコンサルタントや traced_plan 関数に使用されます。

このテーブルには sa_diagnostic_internalvariable と sa_tmp_diagnostic_internalvariable という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
request_id	UNSIGNED BIGINT	内部変数を含む要求 ID。
rowvariable_id	UNSIGNED INT	この値のロー変数のカラム番号。
variable_domain	UNSIGNED SMALLINT	内部変数のデータ型。
variable_name	CHAR(128)	内部変数の名前。
variable_value	LONG NVARCHAR	内部変数の値を表す文字列。

参照

- 「ローカル変数」77 ページ

sa_diagnostic_query テーブル

sa_diagnostic_query テーブルは dbo ユーザーが所有しています。また、特に最適化したコンテキストなど、クエリの最適化情報を格納しています。このテーブルのローは、クエリのオプティマイザーの呼び出しを表します。最適化時にキャプチャされたプランはここに格納されます。

このテーブルの値の一部は、データベースオプション値を反映します。

このテーブルには sa_diagnostic_query と sa_tmp_diagnostic_query という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	クエリまたは要求が発生したときのロギングセッションの ID。
query_id	UNSIGNED BIGINT	クエリをユニークに識別する番号。
statement_id	UNSIGNED BIGINT	クエリで文をユニークに識別する番号。
user_object_id	UNSIGNED BIGINT	このクエリが実行されたときのユーザーのオブジェクト ID。クエリをプロシージャーから実行した場合、これはプロシージャー所有者のユーザー ID になります。
start_time	TIMESTAMP	クエリを最適化した時刻。
cache_size_bytes	UNSIGNED BIGINT	クエリを最適化した時点のキャッシュサイズ (バイト単位)。
optimization_goal	TINYINT	クエリ処理の最適化の対象を、最初のローを迅速に返すこと、または完全な結果セットを返すコストを最小限に抑えることのどちらかに指定します。この値は、optimization_goal データベースオプションの値を反映しています。

カラム名	カラム型	説明
optimization_level	TINYINT	SQL Anywhere クエリオプティマイザーが SQL 文のアクセスプランの検索に費やす作業量を制御します。この値は、optimization_level データベースオプションの値を反映しています。
user_estimates	TINYINT	クエリの述部に含まれるユーザー選択性推定をクエリオプティマイザーが尊重するか無視するかを制御します。この値は、user_estimates データベースオプションの値を反映しています。
optimization_workload	TINYINT	クエリ処理において、更新と読み込みを組み合わせた負荷に対して最適化するか、または大部分が読み込みベースの負荷に対して最適化するかを決定します。この値は、optimization_workload データベースオプションの値を反映しています。
available_requests	TINYINT	クエリ内並列処理のレベルを計算するときに内部的に使用されます。
active_requests	TINYINT	クエリ内並列処理のレベルを計算するときに内部的に使用されます。
max_tasks	TINYINT	クエリ内並列処理のレベルを計算するときに内部的に使用されます。
used_bypass	TINYINT	単純なクエリの回避を使用したかどうか。1 の値は、回避を使用したことを示します。0 の値は、クエリを完全に最適化したことを示します。
estimated_cost_ms	TINYINT	推定コスト (ミリ秒)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	カラム型	説明
plan_explain	LONG VARCHAR	このクエリのテキストプラン表記。
plan_xml	LONG VARCHAR	クエリのグラフィカルプラン表記 (記録された場合)。
sql_rewritten	LONG VARCHAR	最適化を適用した後のクエリのテキスト。最適化のロギングが有効な場合、値はこのカラムにのみ表示されます。

参照

- 「データベースオプション」『SQL Anywhere サーバー データベース管理』
- 「オプティマイザーの仕組み」『SQL Anywhere サーバー SQL の使用法』
- 「optimization_level オプション」『SQL Anywhere サーバー データベース管理』
- 「optimization_workload オプション」『SQL Anywhere サーバー データベース管理』
- 「user_estimates オプション」『SQL Anywhere サーバー データベース管理』

sa_diagnostic_request テーブル

sa_diagnostic_request テーブルは、dbo ユーザーが所有します。また、全要求のマスタテーブルです。要求はクエリ処理に関連するイベントであり、一般的に次のイベントが含まれます。

●イベントの接続または接続解除

●文の実行

●文の準備

●カーソルイベントを開く、または削除

このテーブルには sa_diagnostic_request と sa_tmp_diagnostic_request という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	要求が発生したときのロギングセッション。
request_id	UNSIGNED BIGINT	要求をユニークに識別する番号。

カラム名	カラム型	説明
start_time	TIMESTAMP	イベントが開始した時間。
finish_time	TIMESTAMP	文の実行の場合、文が完了した時間。それ以外の場合は NULL。
duration_ms	UNSIGNED INT	イベントの継続時間 (ミリ秒)。
connection_number	UNSIGNED INT	イベントをトリガーさせた接続の ID。
request_type	UNSIGNED SMALLINT	要求の型。次のような値があります。開始された新しい診断トレーシングセッション、実行された SQL 文、開かれたカーソル、閉じられたカーソル、クライアントの接続、クライアントの切断、チェックポイントです。
statement_id	UNSIGNED BIGINT	イベントが文関連だった場合、トレーシング用途で文に割り当てられた ID。
query_id	UNSIGNED BIGINT	イベントがクエリ関連だった場合、トレーシング用途でクエリに割り当てられた ID。
cursor_id	UNSIGNED BIGINT	イベントがカーソル関連だった場合、トレーシング用途でカーソルに割り当てられた ID。
sql_code	SMALLINT	このテーブルのローは、文、カーソル、クエリに関する操作を表すため、ほとんどの場合、SQL コードを返します。このカラムには返された SQL コードが含まれます。0 の SQL コードが返される場合、カラムには NULL が含まれます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

sa_diagnostic_statement テーブル

sa_diagnostic_statement テーブルは dbo ユーザーが所有します。また、文のテキストを格納します。このテーブルのローは、サーバー側で実行された SQL 文を示します。クライアント要求などの外部ソース、またはプロシージャー、トリガー、ユーザー定義関数などの内部ソースによって、この文が発行された可能性があります。ここには内部的な文のみが 1 セッションにつき 1 つのみ表示されます。

このテーブルには sa_diagnostic_statement と sa_tmp_diagnostic_statement という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	文が送信されたときのロギングセッション。
statement_id	UNSIGNED BIGINT	トレーシング処理用に文に割り当てたユニークな番号。
database_object	UNSIGNED BIGINT	文がプロシージャー、トリガー、または関数に由来する場合、これは ISYSOBJECT システムテーブルに指定されている ID です。
line_number	UNSIGNED SMALLINT	文が複合文の一部である場合、これは複合文内にその文が占める順序位置を反映します。
signature	UNSIGNED INT	内部的に同様のクエリをグループ化するときに使用します。
statement_text	LONG VARCHAR	文のテキスト。

sa_diagnostic_statistics テーブル

sa_diagnostic_statistics テーブルは、dbo ユーザーが所有します。また、サーバーで維持されているパフォーマンスカウンターの履歴が含まれます。各ローは、特定時点の特定パフォーマンスカウンターの値を表します。

このテーブルには sa_diagnostic_statistics と sa_tmp_diagnostic_statistics という 2 つのバージョンがあります。

カラム

カラム名	カラム型	説明
logging_session_id	UNSIGNED INT	診断情報が収集されたロギングセッションを一意に識別する番号。
"time"	TIMESTAMP	パフォーマンスカウンター値をキャプチャした時刻。
counter_id	UNSIGNED SMALLINT	パフォーマンスカウンターをユニークに識別する番号。この counter_id が PROPERTY_NAME 関数を使用して表すプロパティの名前を取得できます。
type	TINYINT	データベース、サーバー、または接続の統計情報のいずれかを示します。サーバーの場合は 0、データベースの場合は 1、接続の場合は 2、外部データベースの場合は 4 を使用できます。
connection_number	UNSIGNED INT	接続の統計情報の場合、このプロパティをキャプチャする接続番号。拡張データベース統計情報の場合、このプロパティをキャプチャしたファイルのファイル番号。それ以外の場合、値は 0 です。
counter_value	UNSIGNED INT	パフォーマンスカウンターの値。

参照

- 「PROPERTY_NAME 関数 [システム]」307 ページ

sa_diagnostic_tracing_level テーブル

sa_diagnostic_tracing_level テーブルは、dbo ユーザーが所有します。また、このテーブルの各ローは、トレーシングデータベースに送信する診断情報の種類を決定する条件です。ロギングデータの一部が、このテーブルの 1 つ以上のローの条件に適合する場合、対応するデータがロギングされます。

このテーブルのデータは、CONNECT TRACING 文または REFRESH TRACING LEVEL 文を使用して作成されます。

カラム

カラム名	カラム型	説明
id	UNSIGNED INT	内部でのみ使用。
scope	CHAR(32)	以下に示す診断トレーシングの範囲。 ● DATABASE ● ORIGIN ● USER ● CONNECTION_NAME ● CONNECTION_NUMBER ● FUNCTION ● PROCEDURE ● EVENT ● TRIGGER ● TABLE

カラム名	カラム型	説明
identifier	CHAR(128)	スコープの識別子。この値は、指定したスコープに応じて変化します。次に例を示します。 ●*スコープ*が DATABASE の場合、*識別子*が提示されない場合があります。 ●*スコープ*が ORIGIN の場合、*識別子*は Internal または External にする必要があります。 ●*スコープ*が USER の場合、*識別子*はユーザーの ID です。 ●*スコープ*が CONNECTION_NAME または CONNECTION_NUMBER の場合、*識別子*はそれぞれ接続の名前または番号です。 ●*スコープ*が FUNCTION、PROCEDURE、EVENT、TRIGGER、または TABLE の場合、*識別子*はオブジェクトの完全修飾識別子です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	カラム型	説明
trace_type	CHAR(32)	指定したスコープについてトレースするデータ型を以下に示します。 ● VOLATILE_STATISTICS ● NONVOLATILE_STATISTICS ● CONNECTION_STATISTICS ● BLOCKING ● PLANS ● PLANS_WITH_STATISTICS ● STATEMENTS ● STATEMENTS_WITH_VARIABLES ● OPTIMIZATION_LOGGING ● OPTIMIZATION_LOGGING_WITH_PLANS
trace_condition	CHAR(32)	プランにのみ適用されます。また、大規模でコストが高いクエリ、またはオプティマイザーが最適化を選択しなかったクエリのどちらをトレースするかを制御します。使用できる値を以下に示します。 ● NONE または NULL ● SAMPLE_EVERY ● ABSOLUTE_COST ● RELATIVE_COST_DIFFERENCE

カラム名	カラム型	説明
value	UNSIGNED INT	*条件*に関連付けられている値。たとえば、*条件*が SAMPLE_EVERY の場合、*condition_value* は時間 (ミリ秒) を反映した正の整数です。追加の規則は次のとおりです。 ●*条件* が NULL または NONE の場合、*condition_value* はありません。 ●*条件* が ABSOLUTE_COST の場合、*condition_value* は文の実際の合計実行コスト (ミリ秒) を反映します。 ●*条件* が RELATIVE_COST_DIFFERENCE の場合、*condition_value* は推定コストに占める割合として実行コストを反映します。
enabled	BIT	ローを有効にするかどうか。つまり、ローのトレーシング設定を有効にするかどうか。1 の場合は有効にします。0 の場合は無効にします。

参照

- 「診断トレーシング条件」『SQL Anywhere サーバー SQL の使用法』
- 「診断トレーシングのスコープ」『SQL Anywhere サーバー SQL の使用法』
- 「ATTACH TRACING 文」487 ページ
- 「REFRESH TRACING LEVEL 文」872 ページ

その他のテーブル

ここでは、データベース内の Java と SQL Remote が使用するシステムテーブルなど、その他のテーブルについて説明します。

RowGenerator テーブル (dbo)

dbo.RowGenerator テーブルは、255 のローを持つ読み込み専用のテーブルとして提供されています。このテーブルは、小規模な結果セットを作成するクエリと一連の数値を必要とするクエリに有用です。

RowGenerator テーブルは、システムプロシージャーとシステムビューによって使用されます。このテーブルは、どのような方法でも修正できません。

sa_rowgenerator システムプロシージャーでも一定範囲の数値を生成できます。

カラム名	カラム型
row_num	SMALLINT

row_num　1 ～ 255 の値

参照

- 「sa_rowgenerator システムプロシージャー」1163 ページ

Java システムテーブル

Java で使用されるシステムテーブルを次にリストします。矢印は、テーブル間での外部キーの関係を示します。矢印は、外部テーブルからプライマリテーブルへ示されています。

Mobile Link システムテーブル

Mobile Link システムテーブルの詳細については、「Mobile Link サーバーのシステムテーブル」『Mobile Link サーバー管理』を参照してください。

SQL Remote システムテーブル

SQL Remote システムテーブルの詳細については、「SQL Remote システムテーブル」『SQL Remote』を参照してください。

Ultra Light のシステムテーブル

Ultra Light のシステムテーブルの詳細については、「Ultra Light のシステムテーブル」『Ultra Light データベース管理とリファレンス』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

システムプロシージャー

この項では、SQL Anywhere に付属のシステムプロシージャーについて説明します。sa_get_table_definition など、一部のシステムプロシージャーは関数として実装されています。しかし、それらもシステムプロシージャーと同じコンテキストと方法で使用されるため、システムプロシージャーに含められており、システムプロシージャーと同じような名前が付けられています (sa_xxx)。

SQL Anywhere に含まれるシステムプロシージャーの種類を次に示します。

- システム情報を表形式で表示するためのシステムプロシージャー。
- Web サービスをサポートするための SOAP および HTTP サービスシステムプロシージャー。
- 電子メールを送信するための MAPI および SMTP システムプロシージャー。
- Transact-SQL システムとカタログプロシージャー。

参照

- 「Adaptive Server Enterprise のシステムプロシージャーとカタログプロシージャー」1033 ページ

システムプロシージャーとファンクションの詳細の表示

データベース管理者は、Sybase Central からシステムプロシージャーとファンクションの定義にアクセスできます。

前提条件

DBA 権限。

内容と備考

次のとおりです。

♦ システムプロシージャーとファンクションの詳細の表示

1. SQL Anywhere 12 プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. データベースを右クリックし、**[所有者フィルタの設定]** をクリックします。
3. **[DBO]** をクリックしてから、**[OK]** をクリックします。

> **注意**
> openxml システムプロシージャーは、dbo ユーザーが所有しない唯一のプロシージャーです。Sybase Central の openxml に関する情報は、表示できません。

4. 左ウィンドウ枠で、**[プロシージャーとファンクション]** をダブルクリックします。
5. 左ウィンドウ枠でプロシージャーを選択し、右ウィンドウ枠で **[SQL]** タブをクリックします。

結果

プロシージャーまたはファンクションの定義が **[SQL]** タブに表示されます。

次の手順

なし。

Web サービスシステムプロシージャー

次のシステムプロシージャーは、Web サービスとともに使用します。

- 「sa_http_header_info システムプロシージャー」
- 「sa_http_php_page システムプロシージャー」
- 「sa_http_php_page_interpreted システムプロシージャー」
- 「sa_http_variable_info システムプロシージャー」
- 「sa_set_http_header システムプロシージャー」
- 「sa_set_http_option システムプロシージャー」
- 「sa_set_soap_header システムプロシージャー」

参照

- 「Web サービス関数」 147 ページ
- 「HTTP Web サーバーとしての SQL Anywhere」『SQL Anywhere サーバー プログラミング』
- 「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

MAPI プロシージャーと SMTP プロシージャー

SQL Anywhere には、Microsoft Messaging API 標準 (MAPI) または Internet 標準のメール転送プロトコル (SMTP) を使用して電子メールを送信するためのシステムプロシージャーが用意されています。これらのシステムプロシージャーは、拡張システムプロシージャーとして実装されます。各プロシージャーは、外部 DLL の関数を呼び出します。

これらのプロシージャーは、dbo ユーザー ID によって所有されています。ユーザーがこれらのプロシージャーを使用するには、EXECUTE パーミッションが必要です。ただし、すでに DBA 権限がある場合は必要ありません。

MAPI または SMTP システムプロシージャーを使用するには、データベースサーバーコンピューターから MAPI または SMTP E-mail システムにアクセスできるようにします。

MAPI システムプロシージャーと SMTP システムプロシージャーは、次のとおりです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **xp_startmail**　MAPI メッセージシステムにログオンして、指定したメールアカウントでメールセッションを開始する。
- **xp_startsmtp**　SMTP メッセージシステムにログオンして、指定したメールアカウントでメールセッションを開始する。
- **xp_sendmail**　指定したユーザーにメールメッセージを送信する。
- **xp_stopmail**　MAPI メールセッションを閉じる。
- **xp_stopsmtp**　SMTP メールセッションを閉じる。

例

次のプロシージャーは、バックアップが完了したことをユーザーグループに通知します。

```
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;
```

参照

- 「xp_startmail システムプロシージャー」1246 ページ
- 「xp_startsmtp システムプロシージャー」1247 ページ
- 「xp_sendmail システムプロシージャー」1241 ページ
- 「xp_stopmail システムプロシージャー」1249 ページ
- 「xp_stopsmtp システムプロシージャー」1249 ページ

MAPI と SMTP 用のシステムプロシージャーのリターンコード

MAPI システムプロシージャーまたは SMTP システムプロシージャーによって、次のコードが返される場合があります。

リターンコード	意味
0	成功
2	xp_startmail または xp_startsmtp が失敗
3	xp_stopmail または xp_stopsmtp が失敗
5	xp_sendmail 失敗

リターンコード	意味
12	添付書類紛失
15	メモリ不足
20	認識できない受信者
25	メールセッション開始失敗

また、MAPI システムプロシージャーによって、次のコードが返される場合があります。

リターンコード	意味
11	受信者不明確
13	ディスクが満杯
14	失敗
16	無効なセッション
17	テキストサイズ過大
18	ファイル数過多
19	受信者数過多
21	ログイン失敗
22	セッション数過多
23	ユーザーアボート
24	MAPI がない

また、SMTP システムプロシージャーによって、次のコードが返される場合があります。

リターンコード	意味
100	ソケットエラー
101	ソケットのタイムアウト
102	SMTP サーバーのホスト名を解決できない
103	SMTP サーバーに接続できない

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

リターンコード	意味
104	サーバーエラー。応答が理解されなかった (たとえば、メッセージの書式が誤っている場合やサーバーが SMTP ではない場合など) このエラーは、xp_startsmtp に対して smtp_auth_username と smtp_auth_password を指定し、サーバーで SMTP 認証機能がサポートされていない場合にも返される
105	TLS エラーが発生した

Adaptive Server Enterprise のシステムプロシージャーとカタログプロシージャー

Adaptive Server Enterprise には、システムプロシージャーとカタログプロシージャーがあります。これらは、多くの管理用の関数を実行し、システム情報を取得します。システムプロシージャーは、システムテーブルからのテーブル取得、システムテーブルの更新、カタログストアドプロシージャーがタブ形式のシステムテーブルから情報を取得する組み込みのストアドプロシージャーです。

SQL Anywhere では、このプロシージャーの一部がサポートされています。ただし、このプロシージャーの使用の詳細については、Adaptive Server Enterprise のマニュアルを参照してください。

Adaptive Server Enterprise システムプロシージャー

次のリストでは、SQL Anywhere に提供されている Adaptive Server Enterprise のシステムプロシージャーについて説明します。

これらのプロシージャーは Adaptive Server Enterprise や Adaptive Server IQ バージョン 12 以前と同様の機能を実行しますが、まったく同じというわけではありません。これらのプロシージャーを使用する既存のスクリプトがある場合は、プロシージャーを調べてください。ストアドプロシージャーのテキストを参照するには、Sybase Central でプロシージャーを開くか、または Interactive SQL から次のコマンドを実行します。

```
sp_helptext 'dbo.procedure_name'
```

Interactive SQL 出力の全テキストを表示するには、出力の表示幅を設定し直す必要があります。設定するには、**[ツール]** » **[オプション]** » **[SQL Anywhere]** » **[トランケーションの長さ]** をクリックし、新しい値を入力します。

システムプロシージャー名	説明
sp_addgroup	データベースにグループを追加する
sp_addlogin	データベースに新規ログイン ID を追加する
sp_addmessage	ストアドプロシージャーの PRINT と RAISERROR の呼び出しで使用され、ISYSUSERMESSAGE にユーザー定義メッセージを追加する
sp_addtype	ユーザー定義データ型を作成する
sp_adduser	データベースに新規ユーザー ID を追加する
sp_changegroup	ユーザーグループを変更、またはグループにユーザーを追加する
sp_dropgroup	データベースからグループを削除する
sp_droplogin	データベースからログイン ID を削除する
sp_dropmessage	ユーザー定義メッセージを削除する
sp_droptype	ユーザー定義データ型を削除する
sp_dropuser	データベースからユーザー ID を削除する
sp_getmessage	PRINT 文と RAISERROR 文で使用され、ISYSUSERMESSAGE から格納されたメッセージ文字列を取り出す
sp_helptext	システムプロシージャー、トリガー、またはビューのテキストを表示する
sp_password	ユーザー ID のパスワードを追加または変更する

Adaptive Server Enterprise カタログプロシージャー

SQL Anywhere は、Adaptive Server Enterprise カタログプロシージャーのサブセットを実装します。次に、実装されたカタログプロシージャーを示します。

カタログプロシージャー名	説明
sp_column_privileges	サポート対象外

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カタログプロシージャー名	説明
sp_columns	指定したカラムのデータ型を返す
sp_fkeys	指定したテーブルの外部キー情報を返す
sp_pkeys	指定したテーブルのプライマリキー情報を返す
sp_special_columns	指定したテーブルのローをユニークに識別するのに最適な一連のカラムを返す
sp_sproc_columns	ストアドプロシージャーの入力パラメーターとリターンパラメーターの情報を返す
sp_statistics	テーブルとそのインデックスの情報を返す
sp_stored_procedures	1 つ以上のストアドプロシージャーの情報を返す
sp_tables	指定したテーブルの FROM 句に置くことができるオブジェクトのリストを返す

システムプロシージャーのアルファベット順リスト

システムプロシージャーは、openxml システムプロシージャーを除き、ユーザー ID dbo が所有者です。これらのプロシージャーのなかには、内部システムで使用するためのものがあります。この項では、システムと内部での使用が目的のプロシージャー以外について説明します。Windows Mobile では、外部関数を呼び出すことができません。

openxml システムプロシージャー

XML ドキュメントから結果セットを生成します。

構文 1

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

構文 2

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 }

引数

- **WITH 句**　結果セットのスキーマと、結果セットの各カラムに対して値を見つける方法を指定します。WITH 句の *xpath* 引数は、2 番目の引数の *xpath* に対する一致と相対的に一致します。WITH 句の式が複数のノードと一致した場合、ドキュメントの順序における最初のノードだけが使用されます。ノードがテキストノードではない場合、テキストノードの下位ノードをすべて追加することにより結果を検索します。WITH 句の式がどのノードにも一致しない場合は、そのローのカラムは NULL になります。

  WITH 句内の *xpath* 引数には、リテラル文字列または変数を指定できます。

  openxml の WITH 句構文は、ストアドプロシージャーから選択する際に使用する構文と似ています。

- **USING FILE | USING VALUE**　USING FILE 句は、ファイルからデータをロードするときに使用します。USING FILE 句を使用するには、DBA または READFILE 権限が必要です。

  USING VALUE 句は、CHAR 型、NCHAR 型、BINARY 型、または LONG BINARY 型の式や、BLOB 文字列からデータをロードするときに使用します。

  - ***xml-data***　結果セットのベースとなる XML。定数、変数、カラムなど、任意の文字列式が使用できます。

    出力に NCHAR カラムが存在する場合、*xml-data* は NCHAR エンコードで直接解析されます。また、xpath 引数と namespaces 引数も NCHAR エンコードで変換され、解析されます。

  - ***xpath***　XPath クエリを含む文字列。XPath を使用すると、問い合わせる XML ドキュメントの構造を記述するパターンを指定できます。この引数にある XPath パターンによって、XML ドキュメントからノードが選択されます。2 番目の *xpath* 引数の XPath クエリに一致する各ノードが、テーブルにローを 1 つずつ生成します。

    WITH 句の *xpath* 引数に指定できるのは、メタプロパティのみです。メタプロパティは、属性のように XPath クエリ内でアクセスされます。*namespaces* が指定されていない場合、デフォルトでプレフィクス mp が Uniform Resource Identifier (URI) urn:ianywhere-com:sa-xpath-metaprop にバインドされます。*namespaces* が指定された場合、この URI は、クエリのメタプロパティにアクセスするために mp または他のプレフィクスにバインドされます。メタプロパティ名の大文字と小文字は区別されます。openxml 文は、次のメタプロパティをサポートします。

    - **@mp:id**　XML ドキュメント内のユニークなノードの ID を返します。データベースサーバーが再起動されると、ドキュメント内の指定されたノードの ID が変更される場合もあります。このメタプロパティの値は、ドキュメントの順序で増えていきます。

    - **@mp:localname**　ノードの名前のローカル部分を返します。ノードに名前がない場合は NULL を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **@mp:prefix**　ノードの名前のプレフィクス部分を返します。ノードに名前がない場合または名前にプレフィクスが付いていない場合は NULL を返します。
- **@mp:namespaceuri**　ノードが含まれているネームスペースの URI を返します。ノードがネームスペースに含まれていない場合は NULL を返します。
- **@mp:xmltext**　XML ドキュメントのサブツリーを XML 形式で返します。たとえば、内部ノードを照合する場合、このメタプロパティを使用して、下位のテキストノードの連結値ではなく、XML 文字列を返します。

○ ***flags***　WITH 句に XPath クエリが指定されていない場合、XML データと結果セットの間で使用されるマッピングを示します。*flags* パラメーターが指定されない場合、デフォルトで結果セットのカラムに属性がマッピングされます。*flags* パラメーターには、次のいずれかの値を指定します。

値	説明
1	XML 属性が結果セットのカラムにマッピングされます (デフォルト)。
2	XML 要素が結果セットのカラムにマッピングされます。

○ ***namespace-declaration***　XML ドキュメント。クエリのスコープ内のネームスペースは、ドキュメントのルート要素から取得されます。ネームスペースが指定されている場合は、すべての *xpath* 引数が指定されていても、必ず *flags* 引数を含めます。

- ***column-name***　結果セットのカラムの名前。
- ***column-type***　結果セットのカラムのデータ型。データ型は、XML ドキュメントから選択した値と互換性がなければなりません。
- **OPTION 句**　OPTION 句は、エスケープ文字、デリミター、エンコードなど、入力ファイルに使用する解析オプションを指定するときに使用します。

○ **ENCODING 句**　ENCODING 句では、ファイルの読み込みに使用されるエンコードを指定できます。

ENCODING 句を指定しない場合、値が CHAR 型または BINARY 型のときはデータベース文字セット (db_charset) が、値が NCHAR 型のときは NCHAR データベース文字セット (nchar_charset) が値のエンコードと見なされます。

○ **BYTE ORDER MARK 句**　BYTE ORDER MARK 句は、エンコード内にバイトオーダーマーク (BOM) があるかどうかを制御するときに指定します。デフォルトでは、このオプションは ON です。この場合、サーバーはデータの最初でバイトオーダーマーク (BOM) を検索して解釈できます。BYTE ORDER MARK が OFF の場合、サーバーは BOM を検索しません。

入力データがエンコードされている場合は、BYTE ORDER MARK 句を指定してください。

ENCODING 句が指定されている場合

●BYTE ORDER MARK オプションが ON のときに、UTF-16BE または UTF-16LE などのエンディアンを持つ UTF-16 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの検証に使用されます。間違ったエンディアンを指定すると、エラーが返されます。

●BYTE ORDER MARK オプションが ON のときに、明示的なエンディアンのない UTF-16 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は、データのエンディアンの確認に使用されます。BOM がない場合は、オペレーティングシステムのエンディアンであると想定されます。

●BYTE ORDER MARK オプションが ON のときに UTF-8 エンコードを指定すると、データベースサーバーはデータの最初で BOM を検索します。BOM がある場合は無視されます。

ENCODING 句が指定されていない場合

●ENCODING 句を指定しないで BYTE ORDER MARK オプションを ON にすると、サーバーは入力データの最初で BOM を検索します。BOM が見つかると、BOM のエンコード (UTF-16BE、UTF-16LE、または UTF-8) に基づいてソースエンコードが自動的に選択され、BOM はロードするデータの一部であるとは見なされなくなります。

●ENCODING 句を指定せず、BYTE ORDER MARK オプションが OFF であるか、または BOM が入力データの先頭で見つからない場合は、データベースの CHAR エンコードが使用されます。

備考

openxml システムプロシージャーは、*xml-data* を解析し、結果をツリーとして雛型化します。ツリーには、要素、属性、テキストノード、その他の XML 構成要素ごとに個別のノードがあります。openxml システムプロシージャーに指定される XPath クエリは、ツリーからノードを選択する際に使用され、選択されたノードはその後、結果セットにマッピングされます。

openxml システムプロシージャーで使用される XML パーサーは妥当性が検証されず、外部 DTD サブセットまたは外部パラメーターのエンティティを読み取りません。

カラム式に複数の一致がある場合、ドキュメントの順序 (解析される前の元の XML ドキュメントの順序) における最初の一致が使用されます。一致するノードがない場合は、NULL が返されます。内部ノードが選択される場合、結果は連結された内部ノードの下位のテキストノードすべてになります。

データ型が BINARY、LONG BINARY、IMAGE、VARBINARY のカラムは、base64 エンコード形式と見なされ、自動的に復号化されます。FOR XML 句を使用して XML を生成した場合、これらのタイプは base64 エンコードであり、openxml システムプロシージャーを使用して復号化できます。

openxml システムプロシージャーは、XPath 構文のサブセットを次のようにサポートしています。

●子、自分、属性、子孫、子孫と自分、親は、完全にサポートされています。

- 省略形または省略形ではない構文のどちらも、サポートされるすべての機能に使用できます。たとえば、'a' は 'child::a' に等しく、'..' は 'parent::node()' と同じです。
- 名前のテストにワイルドカードが使用できます。たとえば 'a/*/b' のようになります。
- サポートされるテストの種類は、node()、text()、processing-instruction()、comment() です。
- フォーム *expr1*[*expr2*] and *expr1*[*expr2*="*string*"] の修飾子を使用できます。この *expr2* はサポートされている XPath 式です。*expr2* が 1 つ以上のノードと一致する場合、修飾子は TRUE と評価されます。たとえば、'a[b]' は、少なくとも 1 つの子 b を持つ a ノードを検索します。a[b="l"] は、テキスト値 l を持つ子 b を少なくとも 1 つ持つ a ノードを検索します。

参照

- 「FOR XML とバイナリデータ」『SQL Anywhere サーバー SQL の使用法』
- 「XPath 式の使用」『SQL Anywhere サーバー SQL の使用法』
- 「SQL データ型」85 ページ
- 「openxml を使用した XML のインポート」『SQL Anywhere サーバー SQL の使用法』
- 「サポートされている文字セット」『SQL Anywhere サーバー データベース管理』
- 「FROM 句」758 ページ
- XPath クエリ言語：http://www.w3.org/TR/xpath。

例

次のクエリは、openxml システムプロシージャーへの最初の引数として指定された XML ドキュメントから結果セットを生成します。

```
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');
```

このクエリは、次の結果を生成します。

ProductName	ProductID
Tee Shirt	301
Baseball Cap	401

次の例では、最初の <ProductType> 要素にエンティティがあります。クエリを実行すると、このノードは、Tee、&、Sweater、Set の 4 つの子がある要素として解析されます。結果セット内で子を連結するには、. を使用します。

```
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');
```

このクエリは、次の結果を生成します。

ProductName	ProductID
Tee Shirt & Sweater Set	301
Baseball Cap	401

次のクエリは、等号述部を使用して、指定された XML ドキュメントから結果セットを生成します。

```
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"]');
```

このクエリは、次の結果セットを生成します。

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

次のクエリは、XPath @attribute 式を使用して結果セットを生成します。

```
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');
```

次のクエリは、上記のクエリで使用されているような XML ドキュメント (ただし、XML 名前空間は使用されています) を操作します。XPath クエリに名前のテストにワイルドカードを使用し、上記のクリックと同じ結果セットを生成する例です。

```
SELECT * FROM openxml( '<Employee  xmlns="http://www.iAnywhere.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');
```

または、名前空間の宣言を指定することもできます。

```
SELECT * FROM openxml( '<Employee  xmlns="http://www.iAnywhere.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.iAnywhere.com/EmployeeDemo"/>' )
```

```
WITH ( EmployeeID INT '@EmployeeID',
       GivenName    CHAR(20) '@GivenName',
       Surname      CHAR(20) '@Surname',
       PhoneNumber  CHAR(10) '@Phone');
```

sa_ansi_standard_packages システムプロシージャー

SQL 文で使用されている、コア以外の SQL 拡張機能に関する情報を返します。

構文

sa_ansi_standard_packages(*standard*, *statement* **)**

引数

- ***standard*** コア拡張機能に使用する規格。SQL:1999 または SQL:2003。
- ***statement*** 評価する SQL 文。

備考

文に使用される、コア以外の拡張機能がない場合、結果セットは空です。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL プリプロセッサー」『SQL Anywhere サーバー プログラミング』
- 「SQLFLAGGER 関数 [その他]」352 ページ
- 「sql_flagger_error_level オプション」『SQL Anywhere サーバー データベース管理』
- 「sql_flagger_warning_level オプション」『SQL Anywhere サーバー データベース管理』

例

次に、sa_ansi_standard_packages システムプロシージャーを呼び出す例を示します。

```
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');
```

このクエリは、次の結果セットを生成します。

package_id	package_name
T612	Advanced OLAP operations
T611	Elementary OLAP operations
F591	Derived tables
T431	Extended grouping capabilities

sa_audit_string システムプロシージャー

文字列をトランザクションログに追加します。

構文

sa_audit_string(*string* **)**

引数

- ***string*** トランザクションログに追加する文字列。

備考

このシステムプロシージャーでは、監査が有効な場合、トランザクションログに格納されている監査情報にコメントが追加されます。この文字列は最大 200 バイトまでです。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「auditing オプション」『SQL Anywhere サーバー データベース管理』
- 「データベースアクティビティの監査」『SQL Anywhere サーバー データベース管理』

例

次の例では、sa_audit_string を使用してコメントをトランザクションログに追加します。

```
CALL sa_audit_string( 'Auditing test' );
```

sa_char_terms システムプロシージャー

CHAR 文字列を単語に分割し、各単語とその位置を 1 つのローとして返します。

構文

sa_char_terms(' *text***' [, '***config-name***' [, '***owner***']])**

引数

- ***text*** 解析する CHAR 文字列。
- ***config-name*** 文字列を処理するときに適用されるテキスト設定オブジェクト。デフォルト値は 'default_char' です。
- ***owner*** 指定されたテキスト設定オブジェクトの所有者。デフォルト値は NULL です。また、所有者が指定されていない場合、または NULL が指定された場合は、現在のユーザーが想定されます。

備考

このシステムプロシージャーを使用すると、テキスト設定オブジェクトの設定が適用されたときに文字列がどのように解釈されるかを確認できます。インデックス処理で、またはクエリ文字列から、どの単語が削除されるかを確かめる場合に便利です。

パーミッション

なし。

関連する動作

なし。

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「sa_nchar_terms システムプロシージャー」1139 ページ

例

次の文は、デフォルトの CHAR テキスト設定オブジェクト default_char を使用して、CHAR 文字列 "It's a work-at-home day!" 内の単語を返します。

```
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

term	position
home	6
day	7

sa_check_commit システムプロシージャー

コミット前に未処理の参照整合性違反がないか確認します。

構文

sa_check_commit(
tname,
keyname
)

引数

- ***tname*** 参照整合性に現在違反しているローのあるテーブルの名前を含む VARCHAR(128) パラメーター
- ***keyname*** 対応する外部キーインデックスの名前を含む VARCHAR(128) パラメーター

備考

データベースオプション wait_for_commit が ON に設定されている場合、または外部キーが CREATE TABLE 文の CHECK ON COMMIT 句を使用して定義されている場合、データベースを更新すると、変更がコミットされる前に違反が解決されていれば、参照整合性違反が発生します。

変更をコミットする前に、sa_check_commit システムプロシージャーを使用して未処理の参照整合性違反があるかどうかを確認できます。

返されたパラメーターは、現在、参照整合性に違反しているローが含まれているテーブルの名前と、対応する外部キーインデックスを表しています。

パーミッション

なし。

関連する動作

なし。

参照

- 「wait_for_commit オプション」『SQL Anywhere サーバー データベース管理』
- 「CREATE TABLE 文」651 ページ

例

次の一連のコマンドを Interactive SQL から実行できます。サンプルデータベースの Departments テーブルからローが削除され、参照整合性違反が発生します。sa_check_commit システムプロシージャーを呼び出して、どのテーブルとキーに未処理の違反があるかを確認します。ROLLBACK で変更をキャンセルします。

```
SET TEMPORARY OPTION wait_for_commit='On'
go
DELETE FROM Departments
go
CREATE VARIABLE tname VARCHAR( 128 );
CREATE VARIABLE keyname VARCHAR( 128 )
go
CALL sa_check_commit( tname, keyname )
go
SELECT tname, keyname
go
ROLLBACK
go
```

sa_clean_database システムプロシージャー

データベースクリーナーを起動し、実行できる期間の最大値を設定します。

構文

sa_clean_database([*duration*])

引数

- ***duration*** クリーン操作の実行が許可される時間 (秒単位)。引数の指定がない場合、または 0 が指定された場合、データベースクリーナーは DB 領域内のすべてのページのクリーンアップが終了するまで実行を続けます。

備考

データベースクリーナーは、デフォルトのスケジュールで実行される内部タスクです。このシステムプロシージャーを使用すると、データベースクリーナーを強制的にただちに実行したり、クリーナーの起動のたびにその実行時間を指定したりできます。データベースの検証中にこのシステムプロシージャーを使用してデータベースクリーナーを起動した場合、検証が完了するまでデータベースクリーナーは実行されません。

要求の一部を後で実行すると、スナップショットアイソレーショントランザクション、インデックス管理、ローの削除などのデータベースタスクをより効率的に実行できます。このような後回しにできるアクティビティには、一般的に、削除済みエントリ、履歴エントリなどの不要なエントリをデータベースページから削除することによるクリーンアップや、アクセス効率を上げるためのデータベースページの再編成などがあります。

こうしたアクティビティを後で実行することにより、現在の要求がより高速に処理されるだけでなく、データベースサーバーの負荷が軽いときにクリーンアップを実行する余地が生まれます。不要なエントリが区別されて、他のトランザクションから認識できなくなりますが、ページ上の領域は使用しているので、いつかは削除する必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

データベースクリーナーは、後回しにされたクリーンアップアクティビティを実行します。データベースクリーナーは、20 秒ごとに実行されるようスケジュールされています。起動されると、データベースを DB 領域全体にわたって順次読み出し、クリーンアップ可能なページを検証してクリーニングし、次のページへ移ります。データベースクリーナーは、データベースサーバーによって自動的に起動された場合は、セルフチューニングプロセスになります。データベースクリーナーによる作業量と実行時間は、DB 領域内の未処理のクリーンアップ可能ページの割合、データベースサーバーの現在のアクティビティ量、データベースクリーナーがすでにクリーニングに費やした時間など、いくつかの要因に左右されます。0.5 秒間実行した後に、クリーナーがサーバーでアクティブな要求を検出した場合、クリーナーは停止し、通常の間隔で実行するよう自らをスケジュールし直します。データベースクリーナーは、サーバーで実行されている要求が他にないときにページ処理を試行することで、サーバーの休止時間を活用します。

データベースクリーナーの統計情報は、次の 4 つのデータベースプロパティから取得できます。

- **CleanablePagesAdded**　クリーンアップが必要なページ数を返します。
- **CleanablePagesCleaned**　クリーンアップ済みのページ数を返します。
- **CleanableRowsAdded**　クリーンアップが必要なロー数を返します。
- **CleanableRowsCleaned**　クリーンアップ済みのロー数を返します。

CleanablePagesAdded 値と CleanablePagesCleaned 値の差は、クリーニングが必要な残りのデータベースページ数を示します。

sa_clean_database システムプロシージャーを使用すると、データベースクリーナーをデータベース内のすべてのページがクリーンアップされるまで実行したり、データベースクリーナーの実行時間の上限を指定したりできます。

データベースクリーナーの動作をさらにカスタマイズするには、クリーンアップが必要なページ数やロー数が指定したしきい値を超えたらデータベースクリーナーを起動するイベントを設定します。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「CREATE EVENT 文」540 ページ
- CleanablePagesAdded データベースプロパティ『SQL Anywhere サーバー データベース管理』
- CleanablePagesCleaned データベースプロパティ『SQL Anywhere サーバー データベース管理』
- CleanableRowsAdded データベースプロパティ『SQL Anywhere サーバー データベース管理』
- CleanableRowsCleaned データベースプロパティ『SQL Anywhere サーバー データベース管理』

例

次の例では、データベースクリーナーの実行時間を 10 秒に設定します。

```
CALL sa_clean_database( 10 );
```

次の例では、データベースクリーナーを毎日実行してデータベース内のすべてのページをクリーンアップするようスケジュールされたイベントを作成します。

```
CREATE EVENT DailyDatabaseCleanup
SCHEDULE
  START TIME '6:00 pm'
  ON ( 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday' )
  HANDLER
    BEGIN
      CALL sa_clean_database( );
    END;
```

次の例では、データベース内のページの 20% 以上でクリーンアップが必要になるとデータベースクリーナーを強制的に実行します。

```
CREATE EVENT PERIODIC_CLEANER
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 dirty 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;
```

sa_column_stats システムプロシージャー

指定したカラムについて多様な統計情報を返します。この統計情報は、オプティマイザーが使用するために管理されているカラムの統計情報とは関係がありません。

構文

sa_column_stats (
[*tab_name*]
[, *col_name*]
[, *tab_owner*]
[, *max_rows*]
)

引数

- ***tab_name*** テーブルの名前を指定する任意の CHAR(128) パラメーター。このパラメーターが指定されていない場合、すべてのテーブルのすべてのカラムについて統計情報が計算されます。

- ***col_name*** 統計情報を計算するカラムを指定する任意の CHAR(128) パラメーター。このパラメーターが指定されていない場合、指定したテーブルのすべてのカラムについて統計情報が計算されます。

- ***tab_owner*** テーブルの所有者を指定する任意の CHAR(128) パラメーター。このパラメーターが指定されていない場合、データベースサーバーは、指定された *tab_name* と一致する最初のテーブルの所有者を使用します。

- ***max_rows*** 計算に使用するロー数を指定する任意の INTEGER パラメーター。このパラメーターが指定されていない場合、デフォルトで 1000 ローが使用されます。0 を指定すると、データベースサーバーはテーブル内のすべてのローに基づいて比率を計算します。

結果セット

table_owner、table_name、column_name という例外はありますが、文字列以外のカラムで結果セットのすべての値は NULL です。また、空のテーブルの場合、num_rows_processed と num_values_compressed は 0 ですが、その他の値はすべて NULL です。

カラム名	データ型	説明
table_owner	CHAR(128)	テーブルの所有者。
table_name	CHAR(128)	テーブル名。
column_name	CHAR(128)	カラムの名前。
num_rows_processed	INTEGER	統計情報を計算するときに読み取るローの総数。
num_values_compressed	INTEGER	圧縮されるカラムの値の数。カラムが圧縮されない場合、この値は 0 です。
avg_compression_ratio	DOUBLE	カラムの圧縮済み値の平均圧縮率 (サイズの縮小率で表示)。カラムが圧縮されない場合、この値は NULL です。
avg_length	DOUBLE	カラムのすべての NOT NULL 文字列の平均長。
stddev_length	DOUBLE	カラムのすべての NOT NULL 文字列の長さの標準偏差。

カラム名	データ型	説明
min_length	INTEGER	カラムのすべての NOT NULL 文字列の最短長。
max_length	INTEGER	カラムの文字列の最大長。
avg_uncompressed_length	DOUBLE	カラムの圧縮されていないすべての NOT NULL 文字列の平均長。
stddev_uncompressed_length	DOUBLE	カラムの圧縮されていないすべての NOT NULL 文字列の長さの標準偏差。
min_uncompressed_length	INTEGER	カラムの圧縮されていないすべての NOT NULL 文字列の最短長。
max_uncompressed_length	INTEGER	カラムの圧縮されていないすべての NOT NULL 文字列の最大長。

備考

データベースサーバーは、指定した所有者、テーブル、カラムの各名前と一致するカラムを決定します。次に、各指定したカラムのデータについて統計情報を計算します。デフォルトで、データベースサーバーはデータの先頭 1000 ローのみを使用します。

avg_compression_ratio の場合は、値を 100 以上にすることはできません。ただし、極めて圧縮しづらいデータが圧縮カラムに挿入された場合は (たとえば、圧縮済みデータなど)、0 未満になることがあります。値が高いほど、高い圧縮率であることを示します。たとえば、戻り値が 80 の場合、圧縮データのサイズは圧縮していないデータのサイズよりも 80% 小さくなっています。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「カラムの圧縮の考慮事項」『SQL Anywhere サーバー データベース管理』

例

次の例では、SELECT 文で sa_column_stats システムプロシージャーを使用して、カラムの圧縮で最も圧縮されるデータベースのカラムを決定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
SELECT * FROM sa_column_stats()
  WHERE num_values_compressed > 0
  ORDER BY avg_compression_ratio desc;
```

次の例では、前の例から選択を絞り込み、bsmith が所有するテーブルのみを検索します。

```
SELECT * FROM sa_column_stats( tab_owner='bsmith' )
  WHERE num_values_compressed > 0
  ORDER BY avg_compression_ratio desc;
```

sa_conn_activity システムプロシージャー

サーバー上の指定したデータベースへの各接続について、最後に作成された SQL 文を返します。

構文

sa_conn_activity([*connidparm*] **)**

引数

- ***connidparm*** 接続 ID 番号を指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
Number	INT	接続の ID 番号を返します。
Name	VARCHAR(255)	現在の接続の名前を返します。 テンポラリ接続名の場合、接続名の先頭に **INT:** が追加されます。
Userid	VARCHAR(255)	接続のユーザー ID を返します。
DBNumber	INT	データベースの ID 番号を返します。
LastReqTime	VARCHAR(255)	指定された接続において最後の要求が開始された時刻を返します。このプロパティは、イベントなどの内部接続の場合は空の文字列を返すことがあります。
LastStatement	LONG VARCHAR	現在の接続で最後に準備された SQL 文を返します。

備考

サーバーが情報の収集を指示された場合、sa_conn_activity システムプロシージャーは、各接続について、最後に作成された SQL 文で構成される結果セットを返します。sa_conn_activity を呼び出す前に、データベースサーバーで文の記録をオンにします。文の記録をオンにするには、データベースサーバーの起動時に -zl オプションを指定するか、次の文を実行します。

```
CALL sa_server_option('RememberLastStatement','ON');
```

このプロシージャーは、データベースサーバーがビジー状態のときに、各接続について作成された最後の SQL 文の情報を取得するのに便利です。この機能は、ロギング要求の代わりとして使用できます。

connidparm を指定しない場合、データベースサーバー上で実行されているすべてのデータベースに対するすべての接続について、情報が返されます。*connidparm* が 0 未満の場合、現在の接続のオプション値が返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「-zl dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「sa_server_option システムプロシージャー」1170 ページ
- Name 接続プロパティ『SQL Anywhere サーバー データベース管理』
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』

sa_conn_compression_info システムプロシージャー

通信の圧縮率を要約します。

構文

sa_conn_compression_info([*connidparm*])

引数

- ***connidparm*** 接続 ID 番号を指定する任意の INTEGER パラメーター。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

結果セット

カラム名	データ型	説明
Type	VARCHAR(20)	この後に続く圧縮統計が 1 つの接続 (Connection) の圧縮統計か、サーバーへの全接続 (Server) の圧縮統計かを識別する文字列を返します。
ConnNumber	INTEGER	接続 ID 番号を表す INTEGER を返します。Type が Server の場合は NULL が返されます。
Compression	VARCHAR(10)	この接続で通信圧縮が有効であるかどうかを示す値として、On または Off を返します。 Type が Server の場合は NULL が返され、Type が Connection の場合は ON／OFF が返されます。
TotalBytes	INTEGER	送受信された実際のバイト数の合計を示す INTEGER を返します。
TotalBytesUnComp	INTEGER	圧縮を無効にした場合に送受信されると推測されるバイト数を示す INTEGER を返します。
CompRate	NUMERIC(5,2)	全体的な圧縮率を示す NUMERIC(5,2) の値を返します。たとえば、0 は圧縮が実行されなかったことを意味します。値が 75 の場合、データの圧縮率は 75% であり、元のサイズの 1/4 に圧縮されていることを意味します。
CompRateSent	NUMERIC(5,2)	クライアントに送信されたデータの圧縮率を示す NUMERIC(5,2) の値を返します。

カラム名	データ型	説明
CompRateReceived	NUMERIC(5,2)	クライアントから受信したデータの圧縮率を示す NUMERIC(5,2) の値を返します。
TotalPackets	INTEGER	送受信された実際のパケット数の合計を示す INTEGER を返します。
TotalPacketsUnComp	INTEGER	圧縮を無効にした場合に送受信されると推測されるパケットの総数を示す INTEGER を返します。
CompPktRate	NUMERIC(5,2)	パケットの全体的な圧縮率を示す NUMERIC(5,2) の値を返します。
CompPktRateSent	NUMERIC(5,2)	クライアントに送信されたパケットの圧縮率を示す NUMERIC(5,2) の値を返します。
CompPktRateReceived	NUMERIC(5,2)	クライアントから受信したパケットの圧縮率を示す NUMERIC(5,2) の値を返します。

備考

接続 ID 番号を指定した場合、sa_conn_compression_info system システムプロシージャーは、指定された接続について、圧縮プロパティで構成される結果セットを返します。*connidparm* を指定しない場合は、このシステムプロシージャーによって、サーバー上のデータベースへの現在の全接続の情報が返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』

例

次の例では、sa_conn_compression_info システムプロシージャーを使用して、サーバーに対する全接続の圧縮プロパティをまとめた結果セットを返します。

```
CALL sa_conn_compression_info( );
```

Type	ConnNumber	Compression	TotalBytes	...
Connection	79	Off	7841	...
Server	(NULL)	(NULL)	2737761	...
...	...	...	...	...

sa_conn_info システムプロシージャー

接続プロパティ情報をレポートします。

構文

sa_conn_info([*connidparm*])

引数

- ***connidparm*** 接続 ID 番号を指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号を返します。
Name	VARCHAR(255)	接続の ID 番号を返します。 Temporary connection names have **INT:** prepended to the connection name.
Userid	VARCHAR(255)	接続のユーザー ID を返します。
DBNumber	INTEGER	データベースの ID 番号を返します。

カラム名	データ型	説明
LastReqTime	VARCHAR(255)	指定された接続において最後の要求が開始された時刻を返します。このプロパティは、イベントなどの内部接続の場合は空の文字列を返すことがあります。
ReqType	VARCHAR(255)	最後の要求のタイプを返します。接続が接続プールによってキャッシュされる場合、その ReqType 値は CONNECT_POOL_CACHE になります。
CommLink	VARCHAR(255)	接続の通信リンクを返します。これは SQL Anywhere がサポートするネットワークプロトコルであり、同一コンピューター接続の場合は local となります。
NodeAddr	VARCHAR(255)	Returns the address of the client in a client/server connection.
ClientPort	INTEGER	クライアントの TCP/IP ポート番号を返します。接続の種類が TCP/IP でない場合は、0 を返します。
ServerPort	INTEGER	データベースサーバーの TCP/IP ポート番号または 0 を返します。
BlockedOn	INTEGER	現在の接続がブロックされていない場合は 0 を返し、ブロックされている場合はロック競合によってブロックされた接続の接続番号を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
LockRowID	UNSIGNED BIGINT	ロックされたローの識別子を返します。 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).
LockIndexID	INTEGER	ロックされたインデックスの識別子を返します。 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. Otherwise, LockTable returns an empty string.
UncommitOps	INTEGER	Returns the number of uncommitted operations.
ParentConnection	INTEGER	データベースオペレーション (データベースのバックアップや作成など) を実行するためにテンポラリ接続を作成した接続の接続 ID を返します。その他のタイプの接続について、このプロパティは NULL を返します。

備考

接続 ID 番号を指定した場合、sa_conn_info システムプロシージャーは、指定された接続について、接続プロパティで構成される結果セットを返します。*connidparm* を指定しない場合は、この

システムプロシージャーによって、サーバー上のデータベースへの現在の全接続の情報が返されます。*connidparm* が 0 未満の場合、現在の接続のオプション値が返されます。

ブロックの場合には、このプロシージャーが返す BlockedOn 値によって、どのユーザーがどのユーザーによってブロックされているかを調べることができます。sa_locks システムプロシージャーを使用して、ブロックされた接続で保持されているロックを表示できます。

これらプロパティに関する詳細については、次のような例を実行できます。

```
SELECT *, DB_NAME( DBNumber ),
  CONNECTION_PROPERTY( 'LastStatement', Number )
  FROM sa_conn_info( );
```

LockRowID の値は、sa_locks プロシージャーの出力でロックを検索するときに使用できます。

LockIndexID の値は、sa_locks プロシージャーの出力でロックを検索するときに使用できます。また、LockIndexID の値は、SYSIDX システムビューを使用して表示できる ISYSIDX システムテーブルのプライマリキーに対応します。

ロックにはそれぞれ関連付けられたテーブルがあるため、LockTable の値を使用して、ロックで接続が待機しているかどうかを明確に判断できます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- Name 接続プロパティ『SQL Anywhere サーバー データベース管理』
- 「sa_locks システムプロシージャー」1111 ページ
- 「SYSIDX システムビュー」1273 ページ

例

次の例では、sa_conn_info システムプロシージャーを使用して、サーバーに対する全接続の接続プロパティをまとめた結果セットを返します。

```
CALL sa_conn_info( );
```

Number	Name	Userid	DBNumber	...
79		DBA	0	...
46	Sybase Central 1	DBA	0	...

Number	Name	Userid	DBNumber	...
...	...	...	...	...

次の例では、sa_conn_info システムプロシージャーを使用して、どの接続がテンポラリ接続を作成したかを示す結果セットを返します。

```
SELECT Number, Name, ParentConnection FROM sa_conn_info();
```

接続 8 がテンポラリ接続を作成し、そのテンポラリ接続によって CREATE DATABASE 文が実行されました。

```
Number      Name              ParentConnection
---------------------------------------------
1000000048  INT: CreateDB       8
9           SQL_DBC_14675af8    (NULL)
8           SQL_DBA_152d5ac0    (NULL)
```

sa_conn_list システムプロシージャー

接続 ID を含む結果セットを返します。

構文

sa_conn_list(
[*connidparm*]
[, *dbidparm*]
)

引数

- ***connidparm*** 接続 ID 番号を指定する任意の INTEGER パラメーター。
- ***dbidparm*** データベース ID 番号を指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号を返します。

備考

パラメーターを指定しない場合、または両方のパラメーターが NULL の場合、データベースサーバーで実行されているすべてのデータベースに対するすべての接続について、接続 ID が返されます。*connidparm* が 0 未満の場合、現在の接続の接続 ID のみが返されます。*connidparm* が NULL で *dbidparm* が 0 未満の場合、現在のデータベースのみの接続 ID を返します。*connidparm* が NULL で、*dbidparm* が NULL ではなく、その値が 0 以上の場合、そのデータベースのみの接続 ID を返します。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_db_list システムプロシージャー」1067 ページ
- 「sa_conn_options システムプロシージャー」1060 ページ

sa_conn_options システムプロシージャー

データベースオプションに対応する接続プロパティのプロパティ情報を返します。

構文

sa_conn_options([*connidparm*] **)**

引数

- ***connidparm*** 接続 ID 番号を指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号を返します。
PropNum	INTEGER	接続プロパティの番号が返されます。
OptionName	VARCHAR(255)	オプション名を返します。
OptionDescription	VARCHAR(255)	オプションの説明を返します。
Value	LONG VARCHAR	オプションの値を返します。

備考

接続 ID を Number として返し、データベースオプションに対応する使用可能な各接続プロパティについて PropNum、OptionName、OptionDescription、Value を返します。

connidparm を指定しない場合は、現在のデータベースへの全接続のオプション値が返されます。*connidparm* が 0 未満の場合、現在の接続のオプション値が返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_db_list システムプロシージャー」1067 ページ
- 「sa_conn_list システムプロシージャー」1059 ページ
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「データベースオプション」『SQL Anywhere サーバー データベース管理』

sa_conn_properties システムプロシージャー

接続プロパティ情報をレポートします。

構文

sa_conn_properties([*connidparm*])

引数

- ***connidparm*** 接続 ID 番号を指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号を返します。
PropNum	INTEGER	接続プロパティの番号が返されます。
PropName	VARCHAR(255)	接続プロパティの名前が返されます。
PropDescription	VARCHAR(255)	接続プロパティの説明が返されます。
Value	LONG VARCHAR	接続プロパティの値が返されます。

備考

接続 ID を Number として返し、使用可能な各接続プロパティについて PropNum、PropName、PropDescription、Value を返します。すべての接続プロパティ、接続に関するデータベースオプション設定、接続に関連する統計情報の値が返されます。NULL 値を持つ有効なプロパティも返されます。

connidparm を指定しない場合は、現在のデータベースへの全接続のプロパティが返されます。*connidparm* が 0 未満の場合、現在の接続のプロパティ値が返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_conn_list システムプロシージャー」1059 ページ
- 「sa_conn_options システムプロシージャー」1060 ページ
- 「システム関数」150 ページ
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』

例

次の例では、sa_conn_properties システムプロシージャーを使用して、全接続の接続プロパティ情報をまとめた結果セットを返します。

```
CALL sa_conn_properties( );
```

Number	PropNum	PropName	...
79	37	CacheHits	...
79	38	CacheRead	...
...	...	...	...

次の例では、sa_conn_properties システムプロシージャーを使用して、CPU 時間の降順で、すべての接続のリストを返します*。

```
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;
```

* この例の提供者は RisingRoad Professional Services (http://www.risingroad.com) の Breck Carter です。

sa_convert_ml_progress_to_timestamp システムプロシージャー

Mobile Link のスクリプト化されたアップロードの場合のみ。スクリプト化されたアップロードの進行状況値を UNSIGNED BIGINT から TIMESTAMP に変換します。

構文

sa_convert_ml_progress_to_timestamp(*progress* **)**

引数

- ***progress*** この関数は、UNSIGNED BIGINT のパラメーターを 1 つ受け取ります。

備考

この関数は、渡される値で表現するタイムスタンプを返します。このプロシージャーは、sa_convert_timestamp_to_ml_progress の反対です。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_convert_timestamp_to_ml_progress システムプロシージャー」1063 ページ
- 「スクリプト化されたアップロード」『Mobile Link クライアント管理』

例

```
SELECT sa_convert_ml_progress_to_timestamp( 3465034611199 );
```

sa_convert_timestamp_to_ml_progress システムプロシージャー

Mobile Link のスクリプト化されたアップロードの場合のみ。スクリプト化されたアップロードの進行状況値を TIMESTAMP から UNSIGNED BIGINT に変換します。

構文

sa_convert_timestamp_to_ml_progress(*t1* **)**

引数

- ***t1*** UNSIGNED BIGINT に変換する進行状況値を指定する TIMESTAMP パラメーター。

備考

この関数は、パラメーターとして渡されるタイムスタンプを表す UNSIGNED BIGINT を返します。このプロシージャーは、sa_convert_ml_progress_to_timestamp の反対です。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_convert_ml_progress_to_timestamp システムプロシージャー」1063 ページ
- 「スクリプト化されたアップロード」『Mobile Link クライアント管理』

例

```
SELECT sa_convert_timestamp_to_ml_progress( CURRENT TIMESTAMP );

SELECT sa_convert_timestamp_to_ml_progress( '1900/01/01 1:00' );
```

sa_copy_cursor_to_temp_table システムプロシージャー

テンポラリテーブルを作成し、そのテーブルにオープンカーソルの結果セットをコピーします。

構文

sa_copy_cursor_to_temp_table(
cursor_name,
table_name
[, *first_row* [, *max_rows*]]
)

引数

- ***cursor_name*** 記述するオープンカーソルの名前。
- ***table_name*** 作成するテンポラリテーブルの名前。
- ***first_row*** テンポラリテーブルにコピーする最初のローの番号。デフォルトは 1 です。
- ***max_rows*** テンポラリテーブルにコピーするローの最大数。デフォルトは 9223372036854775807 (すべてのロー) です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

いくつかの整数カラムのカーソルが存在している場合を想定します。
sa_copy_cursor_to_temp_table では、次の形式の文を使用して、テンポラリテーブルを作成します。

```
BEGIN
 CREATE LOCAL TEMPORARY TABLE TempTab (
    col1 INT,
    col2 INT,
 ...
rownum bigint primary key )
END;
```

sa_copy_cursor_to_temp_table では、カラムに col1、col2 などの名前を付け、名前の重複や、カーソルカラムの名前が適切に定義されていない場合 (たとえば、複雑な式の場合) の問題を回避します。

テンポラリテーブルが作成されると、オープンカーソルの内容が挿入されます。このとき、*first_row* に指定したロー番号まで移動し、*max_rows* に指定した数のローが挿入されます。テンポラリテーブルに内容が挿入された後、カーソルは元のロケーションに再配置されます。

パーミッション

なし。

関連する動作

カーソルからコピーすると、カーソルの独立性設定を使用してローがフェッチされます。これにより、ローがロックされ、カーソルからフェッチする場合と同じ、他の影響が発生する場合があります。

現在の接続の外部で同時に変更が加えられ、カーソルが実体化または独立性設定によって保護されていない場合、プロシージャーが完了した後でカーソルが別のローに配置される可能性があります。たとえば、直前に最新のローとなっていたローが削除された場合は、カーソルが元の位置の後のローに再配置されることがあります。

カーソルからのコピー中にエラーが発生した場合、カーソルは無効な状態になり、カーソルに対するそれ以降の操作はエラーで失敗します。

参照

- 「sa_list_cursors システムプロシージャー」1108 ページ
- 「sa_describe_cursor システムプロシージャー」1071 ページ

例

次のバッチでは、myCursor というカーソルを作成し、Products テーブルからデータをロードします。その後、カーソルが開かれます (OPEN 文)。DROP 文では、myTempTable (すでに存在する場合) を削除します。sa_copy_cursor_to_temp_table を呼び出すと、myTempTable というテンポラリテーブルが作成され、そのテーブルに myCursor の内容がコピーされます。最後に、SELECT 文によって、カーソルからテンポラリテーブルにコピーされたデータが返されます。

```
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
```

sa_db_info システムプロシージャー

データベースのプロパティ情報をレポートします。

構文

sa_db_info([*dbidparm*] **)**

引数

- ***dbidparm*** データベース ID 番号を指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号を返します。
Alias	VARCHAR(255)	データベース名を返します。
File	VARCHAR(255)	パスを含むデータベースルートファイル名を返します。
ConnCount	INTEGER	データベースとの接続の数を返します。プロパティ値には、内部処理に使用されている接続は含まれませんが、イベントと外部環境サポートに使用されている接続は含まれます。
PageSize	INTEGER	データベースのページサイズ (バイト単位) を返します。
LogName	VARCHAR(255)	パスを含むトランザクションログファイル名を返します。

備考

データベース ID を指定した場合、sa_db_info は、指定したデータベースの Number、Alias、File、ConnCount、PageSize、LogName を持つ 1 つのローを返します。

dbidparm を指定しない場合は、すべてのデータベースのプロパティが返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_db_properties システムプロシージャー」1068 ページ
- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』

例

次の文は、サーバーで実行されている各データベースのローを返します。

```
CALL sa_db_info( );
```

Property	Value
Number	0
Alias	demo
File	*C:¥Documents and Settings¥All Users ¥Documents¥SQL Anywhere 12¥Samples ¥demo.db*
ConnCount	1
PageSize	4096
LogName	*C:¥Documents and Settings¥All Users ¥Documents¥SQL Anywhere 12¥Samples ¥demo.log*

sa_db_list システムプロシージャー

データベース ID を返します。

構文

sa_db_list([*dbidparm* **])**

引数

- ***dbidparm*** データベース ID 番号を指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
Number	INTEGER	データベース ID 番号。

備考

dbidparm を指定しない場合、または *dbidparm* が NULL の場合、データベースサーバーで実行されているすべてのデータベースの ID が返されます。*dbidparm* が 0 未満の場合、現在のデータベースの ID のみが返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_conn_list システムプロシージャー」1059 ページ
- 「sa_conn_options システムプロシージャー」1060 ページ
- 「DB_NAME 関数 [システム]」208 ページ

例

次の例は、データベースサーバーで実行中のデータベースの数を返します。

```
SELECT * FROM sa_db_list();
```

次の例は、システムプロシージャー sa_db_list と関数 DB_Name を使用して、データベースサーバー上で実行中のデータベースのリストを返します。

```
SELECT DB_NAME(Number) FROM sa_db_list();
```

sa_db_properties システムプロシージャー

データベースのプロパティ情報をレポートします。

構文

sa_db_properties([*dbidparm*] **)**

引数

- ***dbidparm*** データベース ID 番号を指定する任意の INTEGER パラメーター。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

結果セット

カラム名	データ型	説明
Number	INTEGER	データベース ID 番号。
PropNum	INTEGER	データベースプロパティの番号。
PropName	VARCHAR(255)	データベースプロパティ名。
PropDescription	VARCHAR(255)	データベースプロパティの説明。
Value	LONG VARCHAR	データベースプロパティの値。

備考

データベース ID を指定した場合、sa_db_properties システムプロシージャーはデータベース ID 番号と、使用可能な各データベースプロパティの PropNum、PropName、PropDescription、Value を返します。データベースに関するすべてのデータベースプロパティと統計情報の値が返されます。NULL 値を持つ有効なプロパティも返されます。

dbidparm を指定しない場合は、すべてのデータベースのプロパティが返されます。

「クラウドに関する注意点」テナントデータベース独立性ルールのため、この機能がクラウドで実行された場合、現在のテナントデータベースについての情報のみを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_db_info システムプロシージャー」1066 ページ
- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』

例

次の例では、sa_db_properties() システムプロシージャーを使用して、すべてのデータベースのデータベースプロパティ情報をまとめた結果セットを返します。

```
CALL sa_db_properties( );
```

Number	PropNum	PropName	...
0	0	ConnCount	...

Number	PropNum	PropName	...
0	1	IdleCheck	...
0	2	IdleWrite	...
...	...	...	...

sa_dependent_views システムプロシージャー

指定したテーブルまたはビューのすべての従属ビューリストを返します。

構文

sa_dependent_views('*tbl_name* **' [, ' *owner_name*] ')**

引数

- ***tbl_name*** テーブルまたはビューの名前を指定する CHARACTER パラメーター。
- ***owner_name*** *tbl_name* の所有者を指定する任意の CHARACTER パラメーター。

結果セット

カラム名	データ型	説明
table_id	UNSIGNED INTEGER	テーブルまたはビューのオブジェクト ID。
dep_view_id	UNSIGNED INTEGER	従属ビューのオブジェクト ID。

備考

このプロシージャーを使用して、従属ビューの ID リストを取得します。または、ビュー名などビューに関する詳細情報を返す文でこのプロシージャーを使用できます。

テーブル名と所有者名について指定された条件を満たす既存のテーブルがない場合、エラーは発生しません。また、次の点にも留意してください。

- *tbl_name* の指定は任意で、デフォルト値は NULL です。
- *owner* と *tbl_name* がどちらも NULL の場合は、従属ビューを持つすべてのテーブルに関する情報が返されます。
- *tbl_name* が NULL で *owner* が指定されている場合は、指定された所有者が所有するすべてのテーブルに関する情報が返されます。
- *tbl_name* が指定されていて *owner* が NULL の場合は、指定された名前を持ついずれかのテーブルに関する情報が返されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

デフォルトでは、このプロシージャーの実行に必要なパーミッションはなく、PUBLIC でカタログにアクセスできることが想定されています。DBA は、必要に応じて、ビューやカタログに対するアクセスを制御できます。

パーミッション

なし。

関連する動作

なし。

参照

- 「SYSDEPENDENCY システムビュー」1262 ページ
- 「ビューの依存性」『SQL Anywhere サーバー SQL の使用法』

例

次の例では、sa_dependent_views システムプロシージャーを使用して、SalesOrders テーブルに依存するビューの ID リストを取得します。このプロシージャーは、SalesOrders に対して table_id を、従属ビュー ViewSalesOrders に対して dep_view_id を返します。

```
CALL sa_dependent_views( 'SalesOrders' );
```

次の例では、sa_dependent_views システムプロシージャーを SELECT 文で使用して、SalesOrders テーブルに依存するビューの名前リストを取得します。このプロシージャーは、ViewSalesOrders ビューを返します。

```
SELECT t.table_name FROM SYSTAB t,
sa_dependent_views( 'SalesOrders' ) v
WHERE t.table_id = v.dep_view_id;
```

sa_describe_cursor システムプロシージャー

カーソルのカラムの名前と型に関する情報を記述します。

構文

sa_describe_cursor(*cursor_name* **)**

引数

- ***cursor_name*** 記述するオープンカーソルを識別する VARCHAR(256) 値。

結果セット

カラム名	データ型	説明
column_number	INTEGER	このローが記述するカラムの順序位置 (開始値は 1)。

カラム名	データ型	説明
name	VARCHAR(128)	カラムの名前。
domain_id	SMALLINT	カラムのデータ型。
domain_name	VARCHAR(128)	カラムのデータ型名。
domain_name_with_size	VARCHAR(160)	サイズと精度を含むデータ型名 (CREATE TABLE 関数または CAST 関数で使用されます)。
width	INTEGER	文字列パラメーターでは長さ、数値パラメーターでは精度、その他のデータ型では記憶領域のサイズをバイトで示します。
scale	INTEGER	数値データ型カラムでは小数点以下の桁数、その他のデータ型では 0 を示します。
declared_width	INTEGER	文字列パラメーターでは長さ、数値パラメーターでは精度、その他のデータ型では記憶領域のサイズをバイトで示します。
user_type_id	SMALLINT	該当する場合はユーザー定義のデータ型、それ以外の場合は NULL。
user_type_name	VARCHAR(128)	該当する場合はユーザー定義のデータ型、それ以外の場合は NULL。
correlation_name	VARCHAR(128)	該当する場合は式と関連付けられている相関名、それ以外の場合は NULL。
base_table_id	UNSIGNED INTEGER	式がカラムの場合は table_id、それ以外の場合は NULL。
base_column_id	UNSIGNED INTEGER	式がカラムの場合は column_id、それ以外の場合は NULL。

カラム名	データ型	説明
base_owner_name	VARCHAR(128)	式がカラムの場合は所有者名、それ以外の場合は NULL。
base_table_name	VARCHAR(128)	式がカラムの場合はテーブル名、それ以外の場合は NULL。
base_column_name	VARCHAR(128)	式がカラムの場合はカラム名、それ以外の場合は NULL。
nulls_allowed	BIT	式を NULL (1) にできるかどうかを示すインジケーター。
is_autoincrement	BIT	式が AUTOINCREMENT カラム (1) かどうかを示すインジケーター。
is_key_column	BIT	式が結果セット (1) のキーの一部かどうかを示すインジケーター。詳細については、以下の「備考」の項を参照してください。
is_added_key_column	BIT	式が追加されたキーカラム (1) かどうかを示すインジケーター。詳細については、以下の「備考」の項を参照してください。

備考

sa_describe_cursor システムプロシージャーは、API に依存しないメカニズムによって、カーソルによって返されたカラムの説明を取得します。このシステムプロシージャーは、動的 SQL で動作するストアドプロシージャーを作成する場合に役立ちます。

sa_describe_cursor システムプロシージャーは、CALL 文、または SELECT 文の FROM 句で使用できます。

cursor_name は、現在の接続のオープンカーソルを示すようにしてください。接続のオープンカーソルのリストを取得するには、sa_list_cursors システムプロシージャーを使用します。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_list_cursors システムプロシージャー」1108 ページ
- 「sa_copy_cursor_to_temp_table システムプロシージャー」1064 ページ
- 「SYSDOMAIN システムビュー」1262 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSTABCOL システムビュー」1315 ページ
- 「SYSUSERTYPE システムビュー」1330 ページ
- 「SYSDOMAIN システムビュー」1262 ページ

sa_describe_query システムプロシージャー

1 つのローにクエリの各出力カラムを記述したクエリの結果セットを記述します。

構文

sa_describe_query(
query
[, *add_keys*]
)

引数

- ***query*** 記述されている SQL 文のテキストを指定する LONG VARCHAR パラメーター。
- ***add_keys*** 記述されているクエリの結果セットでローをユニークに識別するカラムのセットを決定するかどうかを指定する任意の BIT パラメーター。デフォルトは 0 です。データベースサーバーはカラムの識別を試行しません。このパラメーターの詳細については、以下の備考部分を参照してください。

結果セット

カラム名	データ型	説明
column_number	INTEGER	このローが記述するカラムの順序位置 (開始値は 1)。
name	VARCHAR(128)	カラムの名前。
domain_id	SMALLINT	カラムのデータ型。
domain_name	VARCHAR(128)	データ型の名前。
domain_name_with_size	VARCHAR(160)	サイズと精度を含むデータ型名 (CREATE TABLE 関数または CAST 関数で使用されます)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
width	INTEGER	文字列パラメーターでは長さ、数値パラメーターでは精度、その他のデータ型では記憶領域のサイズをバイトで示します。
scale	INTEGER	数値データ型カラムでは小数点以下の桁数、その他のデータ型では 0 を示します。
declared_width	INTEGER	文字列パラメーターでは長さ、数値パラメーターでは精度、その他のデータ型では記憶領域のサイズをバイトで示します。
user_type_id	SMALLINT	1 つの場合はユーザー定義のデータ型の type_id、それ以外の場合は NULL。
user_type_name	VARCHAR(128)	1 つの場合はユーザー定義のデータ型の name、それ以外の場合は NULL。
correlation_name	VARCHAR(128)	使用できる場合、式と関連付けられている相関名、それ以外の場合は NULL。
base_table_id	UNSIGNED INTEGER	式がカラムの場合は table_id、それ以外の場合は NULL。
base_column_id	UNSIGNED INTEGER	式がカラムの場合は column_id、それ以外の場合は NULL。
base_owner_name	VARCHAR(128)	式がカラムの場合は所有者名、それ以外の場合は NULL。
base_table_name	VARCHAR(128)	式がカラムの場合はテーブル名、それ以外の場合は NULL。
base_column_name	VARCHAR(128)	式がカラムの場合はカラム名、それ以外の場合は NULL。

カラム名	データ型	説明
nulls_allowed	BIT	式を NULL にできる場合は 1、それ以外の場合は 0 を示すインジケーター。
is_autoincrement	BIT	式が、AUTOINCREMENT になると宣言されているカラムの場合は 1、それ以外の場合は 0 を示すインジケーター。
is_key_column	BIT	式が結果セットのキーの一部である場合は 1、それ以外の場合は 0 を示すインジケーター。詳細については以下の備考部分を参照してください。
is_added_key_column	BIT	式が、追加されたキーカラムの場合は 1、それ以外の場合は 0 を示すインジケーター。詳細については以下の備考部分を参照してください。

備考

sa_describe_query プロシージャーは、API に依存しないメカニズムによって、クエリの結果セットに含まれる式に関する名前とタイプの情報を記述します。

add_keys に 1 が指定されている場合、sa_describe_query プロシージャーはクエリ対象のオブジェクトからカラムセットを検索しようとします。このカラムセットは、記述されるクエリの結果セットのローをユニークに識別するときに使用されます。キーの形式は、クエリ対象のオブジェクトに含まれる 1 つ以上のカラムです。また、クエリで明示的に参照されないカラムがキーに含まれる場合があります。オプティマイザーによってキーが見つかった場合、キーで使用されるカラムは、結果で値が 1 の is_key_column によって識別されます。キーが見つからない場合はエラーが返されます。

キーに含まれていてもクエリで明示的に参照されていないカラムについては、is_added_key_column の値が 1 に設定されます。これはプロシージャーの結果にカラムが追加されていることを示します。それ以外の場合、is_added_key_column の値は 0 です。

add_keys を指定しない場合、または 0 の値を指定する場合、オプティマイザーは結果セットのキーを検索しません。is_key_column と is_added_key_column には NULL が含まれます。

declared_width 値と width 値のどちらもカラムのサイズを記述します。declared_width は、CREATE TABLE 文またはクエリによって定義されたカラムのサイズを記述します。width 値は、クライアントにフェッチされるときのカラムのサイズを示します。クライアントの型表記は、データベースサーバーとは異なることがあります。たとえば、return_date_time_as_string オプションがオンの場合、日付と時間のデータ型は文字列に変換されます。文字列の場合、

character-length セマンティックで宣言されたカラムは、CREATE TABLE のサイズと一致する declared_width 値を持ちます。width 値は、戻り値の文字列を格納するために必要な最大バイト数を示します。次に例を示します。

宣言	width	declared_width
CHAR(10)	10	10
CHAR(10 CHAR)	40	10
TIMESTAMP	タイムスタンプ形式の文字列長によって変わります	8
NUMERIC(10, 3)	10 (精度)	10 (精度)

パーミッション

なし。

関連する動作

なし。

参照

- 「EXPRTYPE 関数 [その他]」229 ページ
- 「文字データ型」85 ページ
- 「return_date_time_as_string オプション」『SQL Anywhere サーバー データベース管理』
- 「SYSDOMAIN システムビュー」1262 ページ
- 「SYSDOMAIN システムビュー」1262 ページ
- 「SYSUSERTYPE システムビュー」1330 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSUSER システムビュー」1328 ページ

例

次の例は、Departments テーブルのすべてのカラムを問い合わせると返される情報を記述します。

```
SELECT *
FROM sa_describe_query( 'SELECT * FROM Departments DEPT' );
```

add_keys パラメーターが指定されなかったため、結果では is_key_column と is_added_key_column の値が NULL になります。

次の例は、Departments テーブルとジョインした Employees テーブルの DepartmentName カラムと Surname カラムを問い合わせることで返される情報を記述します。

```
SELECT  *
FROM  sa_describe_query( 'SELECT DepartmentName, Surname
 FROM Employees E JOIN Departments D ON E.EmployeeID = D.DepartmentHeadId',
 add_keys = 1 );
```

この結果、結果セットのロー 3 と 4 に 1 が示され、クエリの結果セットに含まれるローをユニークに識別するときに必要なカラムが Employees.EmployeeID と Departments.DepartmentID である

ことを示します。また、ロー 3 と 4 の is_added_key_column には 1 が指定されます。これは Employees.EmployeeID と Departments.DepartmentID が、記述されたクエリで明示的に参照されなかったためです。

sa_describe_shapefile システムプロシージャー

ESRI シェイプファイルに含まれるカラムの名前と型を記述します。このシステム機能は、空間データ機能とともに使用します。

構文

sa_describe_shapefile(*shp_filename*
, *srid*
[**,** *encoding*]
)

引数

- ***shp_filename*** ESRI シェイプファイルのロケーションを識別する VARCHAR(512) パラメーター。ファイル名の拡張子は *.shp* にしてください。また、同じディレクトリにある同じベース名の *.dbf* ファイルを関連付けてください。パスは、クライアントアプリケーションではなく、データベースサーバーを基準にした相対パスを指定します。

- ***srid*** シェイプファイル内のジオメトリの SRID を識別する INTEGER パラメーター。カラムに複数の SRID を格納できるようにするには、NULL を指定します。NULL を指定すると、ジオメトリ値に対して実行できる操作が制限されます。

- ***encoding*** シェイプファイルを読み込むときに使用するエンコードを識別する VARCHAR(50) パラメーター。デフォルトのエンコードは ISO-8859-1 です。

結果セット

カラム名	データ型	説明
column_number	INTEGER	このローが記述するカラムの順序位置 (開始値は 1)。
name	VARCHAR(128)	カラムの名前。
domain_name_with_size	VARCHAR(160)	サイズと精度を含むデータ型名 (CREATE TABLE 関数または CAST 関数で使用されます)。

備考

sa_describe_shapefile システムプロシージャーは、ESRI シェイプファイル内のカラムの名前と型を記述するときに使用します。この情報を使用して、LOAD TABLE 文または INPUT 文によってシェイプファイルからデータをロードするテーブルを作成できます。また、このシステムプロ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

シージャーは、OPENSTRING...FORMAT SHAPEFILE の WITH 句を指定することで、シェイプファイルを読み込むときにも使用できます。

パーミッション

-gl データベースオプションが all に設定されている場合は、すべてのユーザーがこのシステムプロシージャーを実行できます。それ以外の場合は、DBA または READFILE 権限が必要です。

参照

- 「LOAD TABLE 文」814 ページ
- 「INPUT 文 [Interactive SQL]」794 ページ
- 「ESRI シェイプファイルのサポート」『SQL Anywhere サーバー 空間データサポート』

例

次の例は、シェイプファイルデータを格納するためのテーブルを作成するときに使用できる文字列を返します。

```
SELECT 'create table esri_load( record_number int primary key, ' ||
   (SELECT list( name || ' ' || domain_name_with_size, ', '
    ORDER BY column_number )
    FROM sa_describe_shapefile( 'c:¥¥esri¥¥shapefile.shp', 1000004326 )
    WHERE column_number > 1 ) || ' )';
```

テーブルを作成したら、次の文を使用してテーブルにシェイプファイルデータをロードできます。

```
LOAD TABLE esri_load USING FILE 'c:¥¥esri¥¥shapefile.shp' FORMAT SHAPEFILE;
```

sa_disable_auditing_type システムプロシージャー

特定のイベントの監査を無効にします。

構文

sa_disable_auditing_type(' *types* **')**

引数

- ***types*** カンマで区切られた文字列を指定する VARCHAR(128) パラメーター。この文字列には、次の 1 つ以上の値が含まれています。
  - **all** すべてのタイプの監査を無効にします。
  - **connect** 成功した接続と失敗した接続の両方の監査を無効にします。
  - **connectFailed** 失敗した接続の監査を無効にします。
  - **DDL** DDL 文の監査を無効にします。
  - **options** パブリックオプションの監査を無効にします。

○ **permission**　パーミッションチェック、ユーザーチェック、SETUSER 文の監査を無効にします。

○ **permissionDenied**　失敗したパーミッションチェックと失敗したユーザーチェックの監査を無効にします。

○ **triggers**　トリガーイベントに応じて監査を無効にします。

備考

sa_disable_auditing_type システムプロシージャーを使用すると、1 つ以上の情報カテゴリの監査を無効にできます。

このオプションを all に設定すると、すべての監査を無効にできます。監査を無効にするには、PUBLIC.auditing オプションを Off に設定する方法もあります。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_enable_auditing_type システムプロシージャー」1082 ページ
- 「データベースアクティビティの監査」『SQL Anywhere サーバー データベース管理』
- 「auditing オプション」『SQL Anywhere サーバー データベース管理』

例

すべての監査を無効にするには、次の文を実行します。

```
CALL sa_disable_auditing_type( 'all' );
```

sa_disk_free_space システムプロシージャー

DB 領域、トランザクションログ、トランザクションログミラー、テンポラリファイルに使用可能な領域に関する情報をレポートします。

構文

sa_disk_free_space([*p_dbspace_name*])

引数

- ***p_dbspace_name***　DB 領域、トランザクションログファイル、トランザクションログミラーファイル、またはテンポラリファイルの名前を指定する VARCHAR(128) パラメーター。

  log、mirror、または temp という DB 領域がある場合は、キーワードの前にアンダースコアを付けます。たとえば、log という DB 領域が存在する場合は、_log を使用してログファイルに関する情報を取得します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SYSTEM を指定すると、メインデータベースファイルに関する情報を取得できます。TEMPORARY または TEMP を指定するとテンポラリファイル、TRANSLOG を指定するとトランザクションログ、TRANSLOGMIRROR を指定するとトランザクションログミラーに関する情報を、それぞれ取得できます。

結果セット

カラム名	データ型	説明
dbspace_name	VARCHAR(128)	これは、DB 領域名、トランザクションログファイル、トランザクションログミラーファイル、またはテンポラリファイルです。
free_space	UNSIGNED BIGINT	ボリューム上の空きバイト数。
total_space	UNSIGNED BIGINT	DB 領域があるドライブで使用可能な合計ディスク領域。

備考

p_dbspace_name パラメーターが指定されていないか NULL の場合、結果セットには DB 領域ごとに 1 つのローと、存在する場合はトランザクションログファイル、トランザクションログミラーファイル、テンポラリファイルごとに 1 つのローが含まれます。*p_dbspace_name* が指定されている場合は、1 つまたは 0 個のローが返ります (このような DB 領域が存在しない場合や、log または mirror が指定されていて、ログファイルまたはミラーファイルがない場合は 0 です)。

パーミッション

DBA 権限。

関連する動作

なし。

例

次の例では、sa_disk_free_space システムプロシージャーを使用して、空き領域についての情報を格納した結果セットを返します。

```
CALL sa_disk_free_space( );
```

dbspace_name	free_space	total_space
system	10952101888	21410402304
translog	10952101888	21410402304
temporary	10952101888	21410402304

参照

- 「事前定義の DB 領域」『SQL Anywhere サーバー データベース管理』

sa_enable_auditing_type システムプロシージャー

監査を有効にし、監査対象のイベントを指定します。

構文

sa_enable_auditing_type('*types*')

引数

- ***types*** カンマで区切られた文字列を指定する VARCHAR(128) パラメーター。この文字列には、次の 1 つ以上の値が含まれています。
  - **all** すべてのタイプの監査を有効にします。
  - **connect** 成功した接続と失敗した接続の両方の監査を有効にします。
  - **connectFailed** 失敗した接続の監査を有効にします。
  - **DDL** DDL 文の監査を有効にします。
  - **options** パブリックオプションの監査を有効にします。
  - **permission** パーミッションチェック、ユーザーチェック、SETUSER 文の監査を有効にします。
  - **permissionDenied** 失敗したパーミッションチェックと失敗したユーザーチェックの監査を有効にします。
  - **triggers** トリガーイベント後の監査を有効にします。

備考

sa_enable_auditing_type は、PUBLIC.auditing オプションと一緒に使用して、特定のタイプの情報の監査を有効にします。

PUBLIC.auditing オプションを On に設定して、監査対象の情報タイプを指定しない場合は、デフォルト設定 (all) が有効になります。この場合、すべてのタイプの監査情報が記録されます。

PUBLIC.auditing オプションを On に設定して、sa_disable_auditing_type を使用してすべてのタイプの監査を無効にすると、監査情報は記録されません。監査を再確立するには、sa_enable_auditing_type を使用して監査対象にする情報のタイプを指定します。

PUBLIC.auditing オプションを Off に設定すると、sa_enable_auditing_type の設定にかかわらず監査情報は記録されません。

パーミッション

DBA 権限。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

なし。

参照

- 「sa_disable_auditing_type システムプロシージャー」1079 ページ
- 「データベースアクティビティの監査」『SQL Anywhere サーバー データベース管理』
- 「auditing オプション」『SQL Anywhere サーバー データベース管理』

例

オプションの監査のみを有効にするには、次のようにします。

```
CALL sa_enable_auditing_type( 'options' );
```

sa_eng_properties システムプロシージャー

データベースサーバーのプロパティ情報をレポートします。

構文

sa_eng_properties( )

結果セット

カラム名	データ型	説明
PropNum	INTEGER	データベースサーバープロパティの番号。
PropName	VARCHAR(255)	データベースサーバープロパティ名。
PropDescription	VARCHAR(255)	データベースサーバープロパティの説明。
Value	LONG VARCHAR	データベースサーバープロパティの値。

備考

使用可能な各サーバープロパティの PropNum、PropName、PropDescription、Value を返します。データベースサーバーに関するすべてのデータベースサーバープロパティと統計情報の値が返されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「システム関数」150 ページ

例

次の文は、使用可能な一連のサーバープロパティを返します。

```
CALL sa_eng_properties( );
```

PropNum	PropName	...
1	IdleWrite	...
2	IdleChkPt	...
...	...	...

sa_external_library_unload システムプロシージャー

外部ライブラリをアンロードします。

構文

sa_external_library_unload(['*lib_name*'])

引数

- ***lib_name*** アンロードするライブラリの名前を指定する任意の LONG VARCHAR パラメーター。ライブラリを指定しない場合は、使用中でない外部ライブラリがすべてアンロードされます。

備考

外部ライブラリを指定したとき、そのライブラリが使用中である場合、またはライブラリがロードされていない場合は、エラーが返されます。パラメーターを指定しない場合、ロードされた外部ライブラリがないときはエラーが返されます。

パーミッション

DBA 権限。

関連する動作

なし。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SQL Anywhere 外部呼び出しインターフェイス」『SQL Anywhere サーバー プログラミング』

例

次の例は、外部ライブラリ myextlib.dll をアンロードします。

```
CALL sa_external_library_unload( 'myextlib.dll' );
```

次の例は、現在使用中でないライブラリをすべてアンロードします。

```
CALL sa_external_library_unload();
```

sa_flush_cache システムプロシージャー

データベースサーバーキャッシュ内の現在のデータベースに対するすべてのページを空にします。

構文

sa_flush_cache( )

備考

データベース管理者は、このプロシージャーを使用して現在のデータベースのデータベースサーバーキャッシュの内容を空にします。パフォーマンスの計測に使用し、同じ結果が繰り返し得られるようにします。

パーミッション

DBA 権限。

関連する動作

なし。

sa_flush_statistics システムプロシージャー

すべてのコストモデル統計をデータベースサーバーキャッシュに保存します。

構文

sa_flush_statistics( )

備考

このプロシージャーを使用して、データベースに現在キャッシュされている現在のコストモデル統計情報をディスクにフラッシュします。sa_get_histogram システムプロシージャーまたはヒストグラムユーティリティ (dbhist) を使用して統計情報を取得できます。このシステムプロシージャーを実行すると、ISYSCOLSTAT システムテーブルが更新されます。通常のオペレーション

では、サーバーによって、ディスクへの統計値の自動書き出しが定期的に行われるため、このプロシージャーを実行する必要はありません。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_get_histogram システムプロシージャー」1089 ページ
- 「SYSCOLSTAT システムビュー」1258 ページ
- 「ヒストグラムユーティリティ (dbhist)」『SQL Anywhere サーバー データベース管理』

sa_get_bits システムプロシージャー

ビット文字列を取得し、文字列内の各ビットのローを返します。デフォルトでは、1 のビット値を持つローのみが返されます。

構文

sa_get_bits(*bit_string* [, *only_on_bits*])

引数

- ***bit_string*** ビットを取得する対象のビット文字列を指定する LONG VARBIT パラメーター。*bit_string* パラメーターが NULL の場合、ローは返されません。

- ***only_on_bits*** オンのビット (値が 1 のビット) を持つローのみを返すかどうかを指定する任意の BIT パラメーター。1 (デフォルト) を指定すると、オンのビットを持つローのみを返します。0 を指定すると、ビット文字列のすべてのビットのローを返します。

結果セット

カラム	データ型	説明
bitnum	UNSIGNED INT	このローで記述されるビットの位置。たとえば、ビット文字列の最初のビットは bitnum が 1 になります。
bit_val	BIT	bitnum 位置にあるビットの値。*only_on_bits* が 1 に設定されている場合、この値は常に 1 です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

sa_get_bits システムプロシージャーは、ビット文字列を復号化し、ビット文字列の各ビットについて 1 つのローを返し、ビットの値を示します。*only_on_bits* が 1 (デフォルト) または NULL の場合、オンのビットに対応するローのみが返されます。最適化により、オンのビットが少ない長いビット文字列を効率的に処理できます。*only_on_bits* が 0 に設定されている場合、ビット文字列の各ビットについて 1 つのローが返されます。

たとえば、文 CALL sa_get_bits('1010') は、ビット文字列の位置 1 と 3 に ON のビットを示す、次の結果セットを返します。

bitnum	bit_val
1	1
3	1

sa_get_bits システムプロシージャーは、ビット文字列を関係に変換するときに使用できます。これは、ビット文字列をテーブルに結合するときや、1 つのバイナリ値としてではなく、結果セットとしてビット文字列を取得するときに使用します。大量に 0 ビットがある場合、これらのビットを取得する必要はないため、ビット文字列を結果セットとして取得する方が効率的です。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_split_list システムプロシージャー」1194 ページ
- 「SET_BIT 関数 [ビット配列]」341 ページ
- 「SET_BITS 関数 [集合]」343 ページ
- 「GET_BIT 関数 [ビット配列]」233 ページ

例

次の例は、sa_get_bits システムプロシージャーを使用して、整数のセットをビット文字列としてコード化してから、ジョインで使用するために復号化する方法を示します。

```
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;
```

sa_get_dtt システムプロシージャー

コストモデルの一部であるディスク転送時間 (DTT: Disk Transfer Time) モデルの現在の値をレポートします。

構文

sa_get_dtt(*file_id* **)**

引数

- ***file_id*** データベースファイル ID を指定する UNSIGNED SMALLINT パラメーター。

備考

file_id は SYSDBSPACE システムビューから取得できます。

このプロシージャーは、内部診断を目的としており、システムテーブル ISYSOPTSTAT からデータを取り出します。

結果セット

カラム名	データ型	説明
BandSize	UNSIGNED INTEGER	ランダムアクセスが行われるディスクのサイズ (ページ単位)。
ReadTime	UNSIGNED INTEGER	1 ページの読み込みの分散コスト (マイクロ秒単位)。
WriteTime	UNSIGNED INTEGER	1 ページの書き込みの分散コスト (マイクロ秒単位)。

パーミッション

なし。

関連する動作

なし。

参照

- 「SYSDBSPACE システムビュー」1260 ページ
- 「SYSOPTSTAT システムビュー」1285 ページ
- 「sa_get_dtt_groupreads システムプロシージャー」1088 ページ

sa_get_dtt_groupreads システムプロシージャー

データベースサーバーでグループ読み込みを発行するコストを評価し、レポートします。

構文

sa_get_dtt_groupreads(*file_id* **)**

引数

- ***file_id*** データベースファイル ID を指定する UNSIGNED SMALLINT パラメーター。

備考

file_id は SYSDBSPACE システムビューから取得できます。sa_get_dtt_groupreads システムプロシージャーによって返される評価は、コストモデルの一部です。この評価を使用して、ソートなどの操作時に適切なグループ読み込みのサイズが選択されます。

このプロシージャーは、内部診断を目的としており、システムテーブル ISYSOPTSTAT からデータを取り出します。SYSOPTSTAT に記録された評価が、指定した DB 領域に存在しない場合、ローは返されません。特定のハードウェアデバイスの評価を調整するには、次の文を実行します。

```
ALTER DATABASE CALIBRATE GROUP READ;
```

結果セット

カラム名	データ型	説明
GroupSize	UNSIGNED INTEGER	ランダムアクセスが行われるディスクのサイズ (ページ単位)。
ReadTime	FLOAT	1 ページの読み込みの分散コスト (マイクロ秒単位)。

パーミッション

なし。

関連する動作

なし。

参照

- 「SYSDBSPACE システムビュー」1260 ページ
- 「SYSOPTSTAT システムビュー」1285 ページ
- 「ALTER DATABASE 文」408 ページ
- 「sa_get_dtt システムプロシージャー」1088 ページ

sa_get_histogram システムプロシージャー

カラムのヒストグラムを取得します。

構文

sa_get_histogram(
col_name,
tbl_name
[, *owner_name*]
)

引数

- ***col_name*** ヒストグラムを取得する対象のカラムを指定する CHAR(128) パラメーター。
- ***tbl_name*** *col_name* があるテーブルを指定する CHAR(128) パラメーター。
- ***owner_name*** *tbl_name* の所有者を指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
StepNumber	SMALLINT	ヒストグラムのバケット番号。最初のバケット (StepNumber = 0) の頻度は、NULL の選択性を示します。
Low	CHAR(128)	バケット内の最も低いカラム値 (その値を含む)。
High	CHAR(128)	バケット内の最も高いカラム値 (その値を含まない)。
Frequency	DOUBLE	バケット内の値の選択性。

備考

このプロシージャーは、内部診断を目的としており、指定したカラムのデータベースサーバーからカラム統計を取り出します。この統計が ISYSCOLSTAT システムテーブルに永続的に格納されている場合、サーバーの実行中に統計はメモリに保持され、定期的に ISYSCOLSTAT に書き込まれます。この結果、sa_get_histogram システムプロシージャーが返す統計情報は、任意の時点で ISYSCOLSTAT から選択して取得した情報とは異なる場合があります。

sa_flush_statistics システムプロシージャーを使用してメモリに保存されている最新の統計情報で ISYSCOLSTAT を手動で更新できます。ただし、この方法は運用環境では推奨されません。診断目的でのみ使用してください。

単一バケットは、結果セット内の Low 値が対応する High 値と等しいことで示されます。

ヒストグラムユーティリティを使用して、ヒストグラムを表示することをおすすめします。

文字列カラムに対する述部の選択性を決定するには、ESTIMATE 関数または ESTIMATE_SOURCE 関数を使用します。文字列カラムに対して、sa_get_histogram とヒストグラムユーティリティが ISYSCOLSTAT システムテーブルから取り出すものはありません。文字列データを取り出そうとすると、エラーが発生します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

たとえば、統計が最近削除された場合は、テーブルまたはマテリアライズドビューに対する統計 (ヒストグラムを含む) が存在しないことがあります。この場合、sa_get_histogram システムプロシージャーの結果セットは空です。テーブルまたはマテリアライズドビューの統計を作成するには、CREATE STATISTICS 文を実行します。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「オプティマイザーの推定と統計」『SQL Anywhere サーバー SQL の使用法』
- 「ヒストグラムユーティリティ (dbhist)」『SQL Anywhere サーバー データベース管理』
- 「SYSCOLSTAT システムビュー」1258 ページ
- 「CREATE STATISTICS 文」643 ページ
- 「sa_flush_statistics システムプロシージャー」1085 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」220 ページ
- 「ESTIMATE 関数 [その他]」219 ページ
- 「ヒストグラムユーティリティ (dbhist)」『SQL Anywhere サーバー データベース管理』

例

たとえば、次の文は、SalesOrderItems テーブルの ProductID カラムのヒストグラムを取り出します。

```
CALL sa_get_histogram( 'ProductID', 'SalesOrderItems' );
```

sa_get_request_profile システムプロシージャー

要求ログを分析し、同様の文の実行時間を判断します。

構文

sa_get_request_profile(
[*filename*
[, *conn_id*
[, *first_file*
[, *num_files*]]]]
)

引数

- ***filename*** 要求ロギングのファイル名を指定する任意の LONG VARCHAR パラメーター。
- ***conn_id*** 接続 ID 番号を指定する任意の UNSIGNED INTEGER パラメーター。
- ***first_file*** 分析する最初の要求ログファイルを指定する任意の INTEGER パラメーター。
- ***num_files*** 分析する要求ログファイルの数を指定する任意の INTEGER パラメーター。

備考

このプロシージャーは、sa_get_request_times を呼び出して要求ログファイルを処理してから、結果をグローバルテンポラリテーブル satmp_request_profile に要約します。このテーブルは、ログからの文、各文の実行回数、合計、平均、最大の各実行時間で構成されています。このテーブルをさまざまな方法でソートして、パフォーマンスの最適化のターゲットを識別できます。

ログファイル (*filename*) を指定しない場合、デフォルトは現在のログファイルです。このファイルは、-zo データベースサーバーオプションを使用して指定するか、または次のように指定します。

```
sa_server_option( 'RequestLogFile', filename )
```

接続 ID を指定すると、ログからの情報のフィルターに使用され、その接続に関する要求だけが取り出されます。

パーミッション

DBA 権限。

関連する動作

オートコミット。

例

次のコマンドは、ファイル *req.out.3*、*req.out.4*、*req.out.5* 内の要求の要求回数を取得します。

```
CALL sa_get_request_profile('req.out',0,3,3);
```

参照

- 「sa_get_request_times システムプロシージャー」1092 ページ
- 「sa_statement_text システムプロシージャー」1196 ページ
- 「sa_server_option システムプロシージャー」1170 ページ
- 「-zo dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

sa_get_request_times システムプロシージャー

要求ログを分析し、文の実行時間を判別します。

構文

sa_get_request_times(*filename*
[, *conn_id*
[, *first_file*
[, *num_files*]]]
)

引数

- ***filename*** 要求ロギングのファイル名を指定する任意の LONG VARCHAR パラメーター。
- ***conn_id*** 接続 ID 番号を指定する任意の UNSIGNED INTEGER パラメーター。

- ***first_file*** 分析する最初のファイルを指定する任意の INTEGER パラメーター。
- ***num_files*** 分析する要求ログファイルの数を指定する任意の INTEGER パラメーター。

備考

このプロシージャーは、指定された要求ログを読み込み、ログから文とその実行時間をグローバルテンポラリテーブル satmp_request_time に移植します。

INSERT や UPDATE のような文の場合、実行時間は単純です。クエリの場合は、文を記述する、カーソルを開く、ローをフェッチする、カーソルを閉じるなどの各操作を含め、文を準備してから削除するまでの時間が計算されます。ほとんどのクエリの場合、これには所要時間が正確に反映されます。カーソルが開いている間に、オペレーター操作やクライアント処理などの他のイベントが発生した場合は、時間値が大きく見えますが、クエリが高コストであることを示しているわけではありません。

このプロシージャーは要求ログ内のホスト変数を認識し、それらの値をグローバルテンポラリテーブル satmp_request_hostvar に移植します。このテンポラリテーブルがない古いデータベースの場合、ホスト変数の値は無視されます。

ログファイルを指定しない場合、デフォルトは現在のログファイルです。このファイルは、-zo を使用してコマンドで指定するか、または次のように指定します。

```
sa_server_option( 'RequestLogFile', filename )
```

接続 ID を指定すると、ログからの情報のフィルターに使用され、その接続に関する要求だけが取り出されます。

パーミッション

DBA 権限。

関連する動作

オートコミット。

例

次のコマンドは、ファイル *req.out.3*、*req.out.4*、*req.out.5* 内の要求の実行時間を取得します。

```
CALL sa_get_request_times('req.out',0,3,3);
```

参照

- 「sa_get_request_profile システムプロシージャー」1091 ページ
- 「sa_statement_text システムプロシージャー」1196 ページ
- 「sa_server_option システムプロシージャー」1170 ページ

sa_get_server_messages システムプロシージャー [旧式]

データベースサーバーメッセージウィンドウの定数を結果セットとして返すことができます。

このシステムプロシージャーは廃止される予定です。代わりに sa_server_messages を使用します。

構文

sa_get_server_messages(*first_line* **)**

引数

- ***first_line*** サーバーメッセージの表示を開始する行番号を指定する INTEGER パラメーター。

結果セット

カラム名	データ型	説明
line_num	INTEGER	サーバーメッセージの行番号。
message_text	VARCHAR(255)	サーバーメッセージのテキスト。
message_time	TIMESTAMP	メッセージの時刻。

備考

このプロシージャーは、表示する最初の行番号を指定する INTEGER パラメーターを受け取り、その行と後続するすべての行のローを返します。開始行が負の場合は、結果セットは使用可能な最初の行から始まります。結果セットには、行番号、メッセージテキスト、メッセージ時刻が含まれます。

パーミッション

なし。

関連する動作

なし。

例

次の例は、sa_get_server_messages システムプロシージャーを使用して、データベースサーバーメッセージウィンドウの 16 行目からの内容を格納した結果セットを返します。

```
CALL sa_get_server_messages( 16 );
```

line_num	message_text	...
16	Windows XP Build 2195 で実行中です。	...
17	2132K のメモリがキャッシュに使用されています。	...

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

line_num	message_text	...
...	...	...

sa_get_table_definition システムプロシージャー

指定されたテーブルとそのインデックス、外部キー、トリガー、付与されたパーミッションの作成に必要な SQL 文を含む LONG VARCHAR 文字列を返します。

構文

sa_get_table_definition(*owner*, *tabname* **)**

引数

- ***owner*** *@tabname* の所有者を指定する CHAR(128) パラメーター。
- ***tabname*** テーブルの名前を指定する CHAR(128) パラメーター。

備考

新しいテーブルを同じ定義を使用して作成するには、sa_get_table_definition システムプロシージャーが返す文字列を、EXECUTE IMMEDIATE 文と LOCATE、SUBSTRING、REPLACE の各関数で使用します。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_split_list システムプロシージャー」1194 ページ
- 「EXECUTE IMMEDIATE 文 [SP]」740 ページ
- 「LOCATE 関数 [文字列]」271 ページ
- 「SUBSTRING 関数 [文字列]」361 ページ
- 「REPLACE 関数 [文字列]」330 ページ

例

次の文は、sa_get_table_definition システムプロシージャーを使用して、Departments テーブルを作成するために必要な SQL 文を含む文字列を表示します。

```
SELECT sa_get_table_definition( 'GROUPO', 'Departments');
```

sa_get_user_status システムプロシージャー

ユーザーの現在のステータスを特定できます。

構文

sa_get_user_status( )

引数

なし。

結果セット

カラム名	データ型	説明
user_id	UNSIGNED INT	ユーザーを識別するユニークな番号。
user_name	CHAR(128)	ユーザーの名前。
connections	INT	このユーザーによる現在の接続数。
failed_logins	UNSIGNED INT	ユーザーがログインしようとして失敗した回数。
last_login_time	TIMESTAMP	ユーザーが最後にログインした時刻。
locked	TINYINT	ユーザーアカウントがロックされているかどうかを示すインジケーター。
reason_locked	LONG VARCHAR	アカウントがロックされた理由。

備考

このプロシージャーは、ユーザーの現在のステータスを示す結果セットを返します。基本的なユーザー情報に加えて、ユーザーがロックアウトされているかどうかを示すカラムや、ロックアウトの理由が格納されたカラムが含まれています。ユーザーは、ポリシーによるロック、パスワードの失効、または失敗した試行回数の過多などの理由によって、ロックアウトされる可能性があります。

DBA 権限のないユーザーは、DBA が所有するカバープロシージャーを作成して実行することによって、ユーザー情報を取得できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

すべてのユーザーに関する情報を表示するには DBA 権限が必要です。DBA 権限のないユーザーが表示できるのは、自分自身の情報です。また、DBA 権限のないユーザーは、DBA が所有するカバープロシージャーを実行することによって、他のユーザーに関する情報を表示できます。

関連する動作

なし。

参照

- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』
- 「新しいログインポリシーの作成」『SQL Anywhere サーバー データベース管理』
- 「ユーザーの作成とログインポリシーの割り当て」『SQL Anywhere サーバー データベース管理』
- 「既存のユーザーへのログインポリシーの割り当て」『SQL Anywhere サーバー データベース管理』
- 「ログインポリシーの変更」『SQL Anywhere サーバー データベース管理』
- 「ログインポリシーの削除」『SQL Anywhere サーバー データベース管理』

例

次の例は、sa_get_user_status システムプロシージャーを使用して、データベースユーザーのステータスを返します。

```
CALL sa_get_user_status;
```

sa_http_header_info システムプロシージャー

HTTP 要求ヘッダー名と値を返します。

構文

sa_http_header_info([*header_parm*] **)**

引数

- ***header_parm*** HTTP ヘッダー名を指定する任意の VARCHAR(255) パラメーター。

結果セット

カラム名	データ型	説明
Name	VARCHAR(255)	HTTP ヘッダー名。
Value	LONG VARCHAR	HTTP ヘッダーの値。

備考

sa_http_header_info システムプロシージャーは、HTTP ヘッダー名と値を返します。任意のパラメーターを使用してヘッダー名を指定しない場合、結果セットにはすべてのヘッダーの値が格納されます。

このプロシージャーは、Web サービス内の HTTP 要求の処理中に呼び出された場合は、空でない結果セットを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「NEXT_HTTP_HEADER 関数 [Web サービス]」290 ページ
- 「HTTP_HEADER 関数 [Web サービス]」249 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

sa_http_php_page システムプロシージャー

ヘッダー、GET または POST データ、プロトコルバージョン、要求 URL、メソッドなど、現在の HTTP 要求のコンテキスト情報を使用して、PHP インタプリタによって解釈される PHP コードを渡した結果を返します。

構文

sa_http_php_page(*php_page* **)**

引数

- ***php_page*** この LONG VARCHAR パラメーターには、開始マーカーと終了マーカー (<? php と ?>) を含めて、解釈される全 PHP コードが含まれます。

備考

このシステムプロシージャーを使用するには、PHP 外部環境をインストールしておく必要があります。

このシステムプロシージャーの所有者は DBO です。ただし、セキュリティの向上のため、sa_http_php_page システムプロシージャーは呼び出し元として実行されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_http_php_page_interpreted システムプロシージャー」1099 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

sa_http_php_page_interpreted システムプロシージャー

ヘッダー、GET または POST データ、プロトコルバージョン、要求 URL、メソッドなど、指定したコンテキスト情報のパラメーターを使用して、PHP インタプリタによって解釈される PHP コードを渡した結果を返します。

構文

sa_http_php_page_interpreted(
php_page,
method,
url,
version,
headers,
request_body
)

引数

- ***php_page*** この LONG VARCHAR パラメーターには、開始マーカーと終了マーカー (<?php と ?>) を含めて、解釈される全 PHP コードが含まれます。

- ***method*** この LONG VARCHAR パラメーターには、HTTP 要求のメソッドが含まれます (たとえば、GET、POST、PUT、またはその他の標準的な要求メソッドのいずれか)。*method* の値は、現在の HTTP 要求の @HttpMethod の値を使用して決定できます。

- ***url*** この LONG VARCHAR パラメーターには、クエリ文字列 (存在する場合) を含めて、完全な HTTP 要求の URL が含まれます。*url* の値は、現在の HTTP 要求の @HttpURI の値を使用して決定できます。

- ***version*** この LONG VARCHAR パラメーターには、HTTP 要求のプロトコルバージョン (たとえば、HTTP/1.1 など) が含まれます。*version* の値は、現在の HTTP 要求の @HttpVersion の値を使用して決定できます。

- ***ヘッダー*** この LONG BINARY パラメーターには、標準の HTTP ヘッダーフォーマット Field-Name: Value¥r¥n の HTTP 要求ヘッダーが含まれます。ヘッダーの値は、次の SELECT 文を使用して、現在の HTTP 要求から取り出すことができます。

```
SELECT LIST( name || ': ' || value, CHAR(13) || CHAR(10) )
  FROM sa_http_header_info();
```

- ***request_body*** この LONG BINARY パラメーターには、HTTP 要求の本文がバイナリ形式で含まれます。*request_body* の値は、HTTP_BODY 関数を使用して、現在の HTTP 要求から取り出すことができます。

備考

このシステムプロシージャーを使用するには、PHP 外部環境をインストールしておく必要があります。

このシステムプロシージャーを Web サービス要求外で使用するには、要求情報を指定する必要があります。PHP コード内に設定されたヘッダーは失われます。

このシステムプロシージャーの所有者は DBO です。ただし、セキュリティの向上のため、sa_http_php_page_interpreted システムプロシージャーは呼び出し元として実行されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「HTTP_BODY 関数 [Web サービス]」246 ページ
- 「PHP 外部環境」『SQL Anywhere サーバー プログラミング』
- 「sa_http_php_page システムプロシージャー」1098 ページ
- 「sa_http_header_info システムプロシージャー」1097 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

sa_http_variable_info システムプロシージャー

HTTP 変数名と値を返します。

構文

sa_http_variable_info([*variable_parm*] **)**

引数

- ***variable_parm*** HTTP 変数名を指定する任意の VARCHAR(255) パラメーター。

結果セット

カラム名	データ型	説明
Name	VARCHAR(255)	HTTP 変数名。
Value	LONG VARCHAR	HTTP 変数の値。

備考

sa_http_variable_info システムプロシージャーは、HTTP 変数名と値を返します。任意のパラメーターを使用して変数名を指定しない場合、結果セットにはすべての変数の値が格納されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

このプロシージャーは、Web サービス内の HTTP 要求の処理中に呼び出された場合は、空でない結果セットを返します。

パーミッション

なし。

関連する動作

なし。

参照

- 「NEXT_HTTP_VARIABLE 関数 [Web サービス]」292 ページ
- 「HTTP_VARIABLE 関数 [Web サービス]」253 ページ
- 「sa_http_header_info システムプロシージャー」1097 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

sa_index_density システムプロシージャー

インデックス内の断片化とスキューの量に関する情報をレポートします。

構文

sa_index_density(
[*tbl_name*
[, *owner_name*]]
)

引数

- ***tbl_name*** テーブル名を指定する任意の CHAR(128) パラメーター。
- ***owner_name*** 所有者名を指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
TableName	CHAR(128)	テーブルの名前。
TableId	UNSIGNED INTEGER	テーブル ID。
IndexName	CHAR(128)	インデックスの名前。

カラム名	データ型	説明
IndexId	UNSIGNED INTEGER	インデックス ID。このカラムには、次のいずれかの値が含まれます。 ● **0** プライマリキーの場合 ● **SYSFKEY.*foreign_key_id*** 外部キーの場合 ● **SYSIDX.*index_id*** その他すべてのインデックスの場合
IndexType	CHAR(4)	インデックスタイプ。このカラムには、次のいずれかの値が含まれます。 ● **PKEY** プライマリキーの場合 ● **FKEY** 外部キーの場合 ● **UI** ユニークインデックスの場合 ● **UC** 一意性制約の場合 ● **NUI** ユニークでないインデックスの場合
LeafPages	UNSIGNED INTEGER	リーフページの数。
Density	DOUBLE	各インデックスページが (平均で) どの程度埋まっているかを示す 0 ～ 1 の間の小数。
Skew	DOUBLE	インデックス内の偏りのレベルを示す数。1 は完全にバランスの取れたインデックスであることを示します。値が大きくなるほどスキューも多くなることを示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

sa_index_density システムプロシージャーは、インデックスの断片化とスキューの程度に関する情報を取得するときに使用します。多数のリーフページを持つインデックスの場合は、Density 値を大きくして不均衡値を小さくすることをおすすめします。

インデックスの密度は、インデックスページの平均の満杯度をパーセントとして反映しています。密度が 0.7 の場合、インデックスページに平均で 70% のインデックスデータが入っていることを示しています。インデックススキューは、平均密度からの一般的な偏差を反映しています。オプティマイザーが選択性推定を行う場合、このスキュー量は重要です。

リーフページ数が少ない場合は、密度とスキューの値を考慮する必要はありません。密度とスキューの値は、リーフページ数が多いときだけ重要です。リーフページ数が多い場合、密度の値が低いと断片化を表し、不均衡値が高いとインデックスが偏っていることを示します。どちらも、パフォーマンス低下の要因になります。REORGANIZE TABLE 文を実行すると、この両方の問題に対処できます。

テーブルを指定しないでこのプロシージャーを呼び出すと、データベース内のすべてのテーブルのすべてのインデックスに関する情報が返されます。

また、**アプリケーションプロファイリングウィザード**を使用すると、インデックスの密度とスキューのレベルが許容範囲内にあるかどうかを判断できます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「インデックスの断片化とスキューの削減」『SQL Anywhere サーバー SQL の使用法』
- 「REORGANIZE TABLE 文」877 ページ
- 「アプリケーションプロファイリングウィザードの使用」『SQL Anywhere サーバー SQL の使用法』

例

次の例は、sa_index_density システムプロシージャーを使用して、データベースのすべてのインデックス内の断片化とスキューの量をまとめた結果セットを返します。

```
CALL sa_index_density( );
```

sa_index_levels システムプロシージャー

インデックス内のレベル数が報告され、パフォーマンスのチューニングに役立てることができます。

構文

sa_index_levels(
[*tbl_name*
[, *owner_name*]]
)

引数

- ***tbl_name*** テーブル名を指定する任意の CHAR(128) パラメーター。
- ***owner_name*** 所有者名を指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
TableName	CHAR(128)	テーブルの名前。
TableId	UNSIGNED INTEGER	テーブル ID。
IndexName	CHAR(128)	インデックスの名前。
IndexId	UNSIGNED INTEGER	インデックス ID。このカラムには、次のいずれかが含まれます。 ● **0** プライマリキーの場合 ● **SYSFKEY.*foreign_key_id*** 外部キーの場合 ● **SYSIDX.*index_id*** その他すべてのインデックスの場合

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
IndexType	CHAR(4)	インデックスタイプ。このカラムには、次のいずれかの値が含まれます。 ● **PKEY** プライマリキーの場合 ● **FKEY** 外部キーの場合 ● **UI** ユニークインデックスの場合 ● **UC** 一意性制約の場合 ● **NUI** ユニークでないインデックスの場合
Levels	INTEGER	インデックス内のレベル数。

備考

インデックスツリー内のレベル数は、そのインデックスを使用しているローに対するアクセスに必要な I/O 操作の数を決定します。レベル数の少ないインデックスの方が、レベル数が多いインデックスよりも効率的です。

このプロシージャーは、テーブル名、テーブル ID、インデックス名、インデックス ID、インデックスタイプ、インデックス内のレベル数で構成される結果セットを返します。

引数が指定されない場合は、データベースにあるすべてのインデックスのレベルが返されます。*tbl_name* のみを指定した場合、そのテーブルのすべてのインデックスのレベルが提示されます。*tbl_name* が NULL であり、*owner_name* が指定された場合は、そのユーザーが所有しているテーブルにあるインデックスのレベルだけが返されます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「CREATE INDEX 文」567 ページ
- 「インデックスを適切に選択するとパフォーマンスに大きな違いが生じる」『SQL Anywhere サーバー SQL の使用法』

例

次の例では、sa_index_levels システムプロシージャーを使用して、Products インデックス内のレベル数を返します。

```
CALL sa_index_levels( );
```

TableName	TableId	IndexName	...	Levels
Products	436	Products	...	1
...	...	...	...	...

sa_install_feature システムプロシージャー

追加の空間機能をインストールします。

構文

sa_install_feature(' *feat_name* ')

引数

- ***feat_name*** インストールする機能を識別する LONG VARCHAR パラメーター。サポートされている機能名は次のとおりです。

値	説明
st_geometry_predefined_uom	新しいデータベースに、デフォルトではインストールされない事前定義の測定単位をインストールします。
st_geometry_predefined_srs	新しいデータベースに、デフォルトではインストールされない事前定義の空間参照系と測定単位をインストールします。
st_geometry_compat_func	一連の空間互換関数をインストールします。これらの関数は、空間メソッドの代わりに使用できます。

機能名の定義は、*%SQLANY12%¥scripts* ディレクトリにある *st_geometry_config.tgz* ファイルに指定されています。このファイルが削除されている場合に、このファイルに依存する機能をインストールしようとすると、エラーが返されます。

備考

feat_name 値を問い合わせて、インストールされる機能を確認できます。たとえば、次のクエリでは、st_geometry_predefined_uom にインストールされる測定単位が返されます。

```
SELECT * FROM dbo.st_geometry_predefined_uom('CREATE');
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

また、上記の例ではパラメーター名も示されるため、WHERE 句を使用して特定の値を問い合わせることができます。たとえば、次の文では chain 測定単位の unit_name パラメーターを問い合わせています。

```
SELECT * FROM dbo.st_geometry_predefined_uom('CREATE') WHERE unit_name='chain';
```

unit_name	unit_type	conversion_factor	...
chain	LINEAR	20.1168	...

次のクエリでは、foot に基づくすべての測定単位が返されます。

```
SELECT * FROM dbo.st_geometry_predefined_uom() WHERE unit_name LIKE '%foot%';
```

インストールされる空間参照系を調べるには、次のクエリを使用します。

```
SELECT * FROM dbo.st_geometry_predefined_srs();
```

次の文は、**organization** と **organization_coordsys_id** 別の空間参照系を問い合わせます。

```
SELECT * FROM dbo.st_geometry_predefined_srs() WHERE organization='EPSG' AND
organization_coordsys_id=2295;
```

パーミッション

DBA または SYS_SPATIAL_ADMIN_ROLE グループのメンバー

参照

- 「空間参照系 (SRS) と空間参照系識別子 (SRID)」『SQL Anywhere サーバー 空間データサポート』
- 「測定単位」『SQL Anywhere サーバー 空間データサポート』
- 「空間互換関数」『SQL Anywhere サーバー 空間データサポート』

例

次の文では、新しいデータベースに、デフォルトではインストールされないすべての事前定義の測定単位をインストールします。

```
CALL sa_install_feature('st_geometry_predefined_uom');
```

sa_java_loaded_classes システムプロシージャー

データベースサーバーによって Java VM に現在ロードされているクラスをリストします。

構文

sa_java_loaded_classes( )

結果セット

カラム名	データ型	説明
class_name	VARCHAR(512)	データベースサーバーによって Java VM に現在ロードされているクラスの名前。

備考

データベースサーバーによって Java VM に現在ロードされているすべての Java クラスの名前を含む結果セットを返します。

このプロシージャーは、不足しているクラスを調べるのに便利です。また、特定のアプリケーションでどの jar ファイルのクラスが使用されているかを調べることもできます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「Java クラスをデータベースにインストールする」『SQL Anywhere サーバー プログラミング』

sa_list_cursors システムプロシージャー

現在の接続で使用されているカーソルのリストを返します。

構文

sa_list_cursors( )

結果セット

カラム名	データ型	説明
handle	UNSIGNED INT	カーソルを識別するユニークなハンドル。
scope	INTEGER	カーソルが開いている呼び出しスタックのスコープ。
cursor_name	VARCHAR(128)	カーソル名。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
is_open	BIT	カーソルが現在開いている (1) かどうかを示すインジケーター。
is_pinned	BIT	カーソルが再利用に備えて現在メモリで固定されている (1) かどうかを示すインジケーター。
fetch_count	UNSIGNED BIGINT	カーソルからフェッチされたローの数。

備考

sa_list_cursors システムプロシージャーは、CALL 文、または SELECT 文の FROM 句で使用できます。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_copy_cursor_to_temp_table システムプロシージャー」1064 ページ
- 「sa_describe_cursor システムプロシージャー」1071 ページ

例

次の例では、接続のオープンカーソルのリストを返します。

```
CALL sa_list_cursors();
```

sa_load_cost_model システムプロシージャー

現在のコストモデルを、指定したファイルに格納されているコストモデルで置換します。

構文

sa_load_cost_model (*file_name* **)**

引数

- ***file_name*** ロードするコストモデルファイルの名前を指定する CHAR(1024) パラメーター。

備考

オプティマイザーは、コストモデルを使用して、クエリの最適なアクセスプランを決定します。データベースサーバーは、各データベースのコストモデルを管理します。データベースのコストモデルは、ALTER DATABASE 文の CALIBRATE SERVER 句を使用して、任意のタイミングで再調整できます。たとえば、データベースを非標準ハードウェアに移動する場合、コストモデルを再調整できます。

sa_load_cost_model システムプロシージャーを使用すると、ファイル (*file_name*) に保存されているコストモデルをロードできます。コストモデルをロードすると、データベースの現在のコストモデルが置換されます。

> **注意**
> sa_unload_cost_model システムプロシージャーは、sa_load_cost_model がロードするファイルに CALIBRATE PARALLEL READ 情報を含めません。

大量に同一ハードウェアのインストールがある場合、sa_load_cost_model システムプロシージャーを使用すると、繰り返しの時間がかかる再調整動作を省略できます。

新規コストモデルをロードする場合、データベースを排他的に使用する必要があります。

コストモデルをロードするときは、類似するハードウェアにあるデータベースのモデルが生成されていないかどうかを検討します。大幅に異なるハードウェアに格納されているデータベースからコストモデルをロードすると、アクセスプランが非効率なため、パフォーマンスが低下する可能性があります。

コストモデルは、sa_unload_cost_model システムプロシージャーを使用してファイルに保存されます。

パーミッション

DBA 権限。

関連する動作

データベースサーバーは新規コストモデルをロードした後に COMMIT を実行します。

参照

- 「ALTER DATABASE 文」408 ページ
- 「sa_unload_cost_model システムプロシージャー」1206 ページ
- 「高度：クエリ最適化」『SQL Anywhere サーバー SQL の使用法』

例

次の例は、costmodel8 というファイルからコストモデルをロードします。

```
CALL sa_load_cost_model( 'costmodel8' );
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

sa_locks システムプロシージャー

データベース内のすべてのロックを表示します。

構文

sa_locks(
[*connection*
[, *creator*
[, *table_name*
[, *max_locks*]]]]
)

引数

- ***connection*** 接続 ID 番号を指定する INTEGER パラメーター。プロシージャーは、指定された接続に関するロック情報のみを返します。デフォルト値は 0 (NULL) で、この場合は全接続の情報が返されます。

- ***creator*** ユーザー ID を指定する CHAR(128) パラメーター。プロシージャーは、指定されたユーザーが所有するテーブルに関する情報のみを返します。creator パラメーターのデフォルト値は NULL です。このパラメーターが NULL に設定されている場合、sa_locks は次の情報を返します。

  ○table_name パラメーターが指定されていない場合、データベースのすべてのテーブルに関するロック情報が返されます。

  ○table_name パラメーターを指定すると、現在のユーザーが作成した、指定した名前を持つテーブルのロック情報が返されます。

- ***table_name*** テーブル名を指定する CHAR(128) パラメーター。プロシージャーは、指定されたテーブルに関するロック情報のみを返します。デフォルト値は NULL で、この場合はすべてのテーブルの情報が返されます。

- ***max_locks*** 情報が返されるロックの最大数を指定する INTEGER パラメーター。デフォルト値は 1000。値が -1 の場合は、すべてのロック情報が返されます。

結果セット

カラム名	データ型	説明
conn_name	VARCHAR(128)	現在の接続の名前。
conn_id	INTEGER	接続 ID 番号。
user_id	CHAR(128)	接続のユーザー ID。

カラム名	データ型	説明
table_type	CHAR(6)	テーブルのタイプ。タイプは、テーブルの場合は BASE、グローバルテンポラリテーブルの場合は GLBTMP、マテリアライズドビューの場合は MVIEW です。
creator	VARCHAR(128)	テーブルの所有者。
table_name	VARCHAR(128)	ロックが保持されているテーブル。
index_id	INTEGER	インデックス ID または NULL。
lock_class	CHAR(8)	ロッククラス。Schema、Row、Table、または Position のいずれか。
lock_duration	CHAR(11)	ロックの継続期間。Transaction、Position、Connection のいずれか。
lock_type	CHAR(9)	ロックの種類 (これはロッククラスとは関係ありません)。
row_identifier	UNSIGNED BIGINT	ローの識別子。8 バイトのロー識別子または NULL です。

備考

sa_locks プロシージャーは、データベース内の全ロックの情報で構成される結果セットを返します。

lock_type カラムの値は、lock_class カラムのロック分類によって変わります。次の値が返されます。

ロッククラス	ロック種類	コメント
Schema	Shared (共有スキーマロック) Exclusive (排他スキーマロック)	スキーマロックの場合、row_identifier とインデックス ID 値は NULL です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ロッククラス	ロック種類	コメント
Row	Read (読み取りロック) Intent (意図的ロック) ReadPK (読み取りロック) Write (書き込みロック) WriteNoPK (書き込みロック) Surrogate (代理ロック)	ローの読み取りロックは、短期のロック (独立性レベル 1 でスキャン) にするか、高い独立性レベルの長期ロックにできます。lock_duration カラムは、読み取りロックがカーソル安定性 (Position) のために短期になるか、COMMIT/ROLLBACK (Transaction) まで保持される長期になるかを指定します。ローのロックは、常に特定のローに保持されます。この 8 バイトのロー識別子は、row_identifier カラムの 64 ビット整数値としてレポートされます。代理ロックは、ローロックの特殊な場合です。代理ロックは代理エントリで保持されます。代理エントリは、参照整合性のチェックが遅延するときに作成されます。 テーブルで作成されるすべての代理エントリについてユニークな代理ロックはありません。ただし、代理ロックは、指定した接続で指定されるテーブルで作成される代理エントリのセットに対応します。row_identifier 値は、代理ロックに関連付けられているテーブルと接続ごとにユニークです。 必要に応じて、ローのキーとキーではない部分を別々にロックできます。1 つの接続で共有 (読み取り) アクセス用にローのキー部分に対する読み取りロックを取得して、他の接続でローのキーではない他のカラムに対する書き込みロックを取得できるようにできます。ローのキーではないカラムを更新しても、その

ロッククラス	ロック種類	コメント
		ローを参照する外部ローの挿入と削除は妨げられません。
Table	Shared (共有テーブルロック) Intent (テーブルロックを更新する意図) Exclusive (排他テーブルロック)	「テーブルロック」『SQL Anywhere サーバー SQL の使用法』を参照してください。
Position	Phantom (幻ロック) Insert (挿入ロック)	通常、位置ロックも特定のローに保持され、そのローの 64 ビットロー識別子は結果セットの row_identifier カラムに表示されます。ただし、row_identifier カラムが NULL の場合、Position ロックは、全体のスキャン (インデックススキャン、または逐次スキャン) に保持されます。

位置ロックは、逐次テーブルスキャンまたはインデックススキャンと関連付けられます。index_id カラムは、位置ロックが逐次スキャンに関連付けられているかどうかを示します。逐次スキャンのために位置ロックを保持する場合、index_id カラムは NULL です。位置ロックが特定のインデックススキャンの結果として維持される場合、そのインデックスのインデックス識別子は、index_id カラムにリストされます。インデックス識別子は、SYSIDX ビューを使用して表示できる ISYSIDX システムテーブルのプライマリキーに対応します。位置ロックがすべてのインデックスのスキャンで保持される場合、インデックス ID 値は -1 です。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「ロックの仕組み」『SQL Anywhere サーバー SQL の使用法』
- 「SYSIDX システムビュー」1273 ページ
- 「ロックできるオブジェクト」『SQL Anywhere サーバー SQL の使用法』
- 「スキーマロック」『SQL Anywhere サーバー SQL の使用法』
- 「挿入時のロック」『SQL Anywhere サーバー SQL の使用法』
- 「ローロック」『SQL Anywhere サーバー SQL の使用法』
- 「位置ロック」『SQL Anywhere サーバー SQL の使用法』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

このシステムプロシージャーの例と返される情報量を増やすためのヒントについては、「ロック情報の取得方法」『SQL Anywhere サーバー SQL の使用法』を参照してください。

sa_make_object システムプロシージャー

ALTER 文を実行する前に、オブジェクトのスケルトンインスタンスが存在することを確認します。

構文

sa_make_object(
objtype,
objname
[, *owner*
[, *tabname*]]
)

objtype:
'procedure'
| **'function'**
| **'view'**
| **'trigger'**
| **'service'**
| **'event'**

引数

- ***objtype*** 作成されるオブジェクトのタイプを指定する CHAR(30) パラメーター。objtype が **'trigger'** の場合、この引数はトリガーが作成されるテーブルの所有者を示します。
- ***objname*** 作成されるオブジェクトの名前を指定する CHAR(128) パラメーター。
- ***owner*** 作成されるオブジェクトの所有者を指定する任意の CHAR(128) パラメーター。デフォルト値は CURRENT USER です。
- ***tabname*** この CHAR(128) パラメーターは、objtype が 'trigger' である場合にのみ必要です。トリガーを作成するテーブルの名前を指定するときに使用します。

備考

このプロシージャーは、データベーススキーマの作成または変更のために繰り返し実行されるスクリプトで使用すると便利です。このようなスクリプトに共通する問題は、最初の実行では CREATE 文を実行する必要がありますが、その後は ALTER 文を実行する必要があることです。このプロシージャーを使用すると、オブジェクトが存在するかどうかを確認するためにシステムビューを問い合わせる必要がなくなります。

使用するには、このプロシージャーの後にオブジェクト定義全体を含む ALTER 文を実行します。

パーミッション

データベースオブジェクトの作成または変更には、RESOURCE 権限が必要です。

関連する動作

オートコミット。

参照

- 「ALTER EVENT 文」417 ページ
- 「ALTER FUNCTION 文」421 ページ
- 「ALTER PROCEDURE 文」433 ページ
- 「ALTER TRIGGER 文」481 ページ
- 「ALTER VIEW 文」484 ページ
- 「ALTER SERVICE 文 [HTTP Web サービス]」441 ページ
- 「ALTER SERVICE 文 [SOAP Web サービス]」446 ページ

例

次の文は、スケルトンプロシージャー定義が作成されていることを確認し、プロシージャーを定義し、プロシージャーに対するパーミッションを付与します。これらの命令が記述されたスクリプトファイルは、データベースに対して繰り返し実行でき、エラーは起こりません。

```
CALL sa_make_object( 'procedure','myproc' );
ALTER PROCEDURE myproc( in p1 INT, in p2 CHAR(30) )
BEGIN
   // ...
END;
GRANT EXECUTE ON myproc TO public;
```

次の例は、sa_make_object システムプロシージャーを使用して、スケルトン Web サービスを追加します。

```
CALL sa_make_object( 'service','my_web_service' );
```

sa_materialized_view_can_be_immediate システムプロシージャー

指定されたマテリアライズドビューを即時ビューとして定義できるかどうかを返します。

構文

sa_materialized_view_can_be_immediate(
view_name
, *owner_name*
)

引数

- ***view_name*** マテリアライズドビューの名前を指定する CHAR(128) パラメーター。*view_name* が NULL の場合、エラーが返されます。

- ***owner_name*** マテリアライズドビューの所有者を指定する CHAR(128) パラメーター。*owner_name* が NULL の場合、エラーが返されます。

備考

指定された手動ビューを即時ビューに変更できるかどうかについては、制限があります。このシステムプロシージャーは、変更が許可されるかどうかを確認するときに使用します。

sa_materialized_view_can_be_immediate システムプロシージャーは、指定されたマテリアライズドビューに関する次の情報を返します。

カラム名	データ型	説明
SQLStateVal	CHAR(6)	返される SQLSTATE。
ErrorMessage	LONG VARCHAR	SQLSTATE に対応するエラーメッセージ。

結果セット内のそれぞれのローが、ビューについて返される 1 つの SQLSTATE に対応します。そのため、マテリアライズドビュー定義が複数の制限に違反している場合は、そのビューの複数のローが結果に格納されます。

このシステムプロシージャーの出力を sa_materialized_view_info システムプロシージャーの出力と結合すると、ビューのステータスに関する情報と、ビューを即時ビューに変更できるかどうかの情報を取得できます。

パーミッション

DBA 権限。または、DBO が所有するプロシージャーに対する実行パーミッション。

参照

- 「高度：マテリアライズドビューの再表示タイプの変更」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューを手動から即時に変更する際の制限」『SQL Anywhere サーバー SQL の使用法』
- 「sa_materialized_view_info システムプロシージャー」1118 ページ

関連する動作

指定したマテリアライズドビューのすべてのメタデータと依存性は、サーバーのキャッシュにロードされます。

例

次の文を実行し、手動ビュー view10 を作成してそのビューをリフレッシュします。

```
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;
```

次のクエリを使用して、ビュー view10 を即時ビューに変更できない理由を検索します。

```
SELECT SQLStateVal AS "SQLstate", ErrorMessage AS Description
  FROM sa_materialized_view_can_be_immediate( 'view10', 'DBA' )
  ORDER BY SQLSTATE;
```

SQLstate	Description
42WC3	「マテリアライズドビュー %1 を即時に変更できません。すでに初期化されています。」『エラーメッセージ』 を参照。
42WCA	「マテリアライズドビュー %1 を即時に変更できません。NULL 入力不可なカラムにユニークなインデックスがありません。」『エラーメッセージ』 を参照。
42WC6	「マテリアライズドビューを即時に変更できません。COUNT(*) が SELECT リストの一部に必要です。」『エラーメッセージ』 を参照。
42WC7	「マテリアライズドビューを即時に変更できません。非集合の NULL 入力不可のカラムにユニークなインデックスがありません」『エラーメッセージ』 を参照。

sa_materialized_view_info システムプロシージャー

指定されたマテリアライズドビューに関する情報を返します。

構文

sa_materialized_view_info(
[*view_name*
[, *owner_name*]]
)

引数

- ***view_name*** 情報を返す対象のマテリアライズドビューの名前を指定する任意の CHAR(128) パラメーター。
- ***owner_name*** マテリアライズドビューの所有者を指定する任意の CHAR(128) パラメーター。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

view_name と *owner_name* のどちらも指定されていない場合、データベースに含まれるすべてのマテリアライズドビューに関する情報が返されます。

owner_name を指定しない場合は、名前が *view_name* であるすべてのマテリアライズドビューに関する情報が返されます。

sa_materialized_view_info システムプロシージャーは、マテリアライズドビューに関する次の情報を返します。

カラム名	データ型	説明
OwnerName	CHAR(128)	ビューの所有者。
ViewName	CHAR(128)	ビューの名前。
Status	CHAR(1)	ビューに関するステータス情報。可能な値は、次のとおりです。 ● **D** 無効 ● **E** 有効

カラム名	データ型	説明
DataStatus	CHAR(1)	ビュー内のデータに関するステータス情報。可能な値は、次のとおりです。 ● **E** 最後にリフレッシュしようとしたときにエラーが発生しました。ビューは有効ですが、初期化されていません。 ● **F** 最後のリフレッシュ以降、基本となるテーブルが変更されていません。ビューは新しいものと見なされます。ビューは有効であり、初期化されています。 ● **N** ビューは初期化されていません。これは、次のいずれかに当てはまるときに発生します。 ○ビューが作成されて以降、リフレッシュされていない ○ビューからデータがトランケートされている ○ビューが無効である ● **S** 最後のリフレッシュ以降、基本となるテーブルが変更されています。ビューは古いものと見なされます。ビューは有効であり、初期化されています。
ViewLastRefreshed	TIMESTAMP	ビューを最後にリフレッシュした時間。ViewLastRefreshed の値が NULL の場合、ビューは初期化されていません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
DataLastModified	TIMESTAMP	古いビューの場合、基本となるデータを修正した最後の時間。 初期化されていないビューまたは古いと見なされないビューの場合、この値は NULL です。

カラム名	データ型	説明
AvailForOptimization	CHAR(1)	オプティマイザーによるビューの使用可能性に関する情報。可能な値は、次のとおりです。 ● **D** オプティマイザーによる使用は無効です。ビューの所有者は、オプティマイザーがビューを使用することを許可していません。 ● **I** ビューの定義が必要な条件を満たしていないなど、何らかの内部的な理由によって、オプティマイザーがビューを使用できません。ただし、所有者は、オプティマイザーがビューを使用することを明示的に禁止していません。 ● **N** リフレッシュが実行されていないか、または失敗したために、ビューにデータがありません。ビューの所有者の許可により、オプティマイザーはビューを使用できますが、ビューが初期化されていません。 ● **O** 現在の接続に互換性のないオプション値があります。オプティマイザーはビューを使用でき、ビューの定義は必要な条件をすべて満たしていますが、現在のオプション設定が、ビューの作成に使用されたオプション設定と互換性がありません。 ● **Y** オプティマイザーはビューを使用できます。ビューの所有者は、オプ

カラム名	データ型	説明
		ティマイザーがビューを使用することを許可しています。また、ビュー定義は、オプティマイザーが使用するために必要なすべての条件を満たしています。
RefreshType	CHAR(1)	ビューのリフレッシュタイプ。可能な値は、次のとおりです。 ● **I** ビューは即時ビューです。即時ビューは、基本となるテーブルのデータへの変更が、マテリアライズドビューのデータに影響を与えるとすぐにリフレッシュされます。 ● **M** ビューは手動ビューです。手動ビューは、たとえば、REFRESH MATERIALIZED VIEW 文を使用するか、または sa_refresh_materialized_views システムプロシージャーを使用して手動でリフレッシュします。

このプロシージャーは、ビューの定義に問題があるために、オプティマイザーによって考慮されないマテリアライズドビューのリストを決定する場合に便利です。このようなマテリアライズドビューの場合、AvailForOptimization 値は I です。

次の表は、AvailForOptimization プロパティがどのように決定されるかを示しています。左のカラムからローを横に読んでいき、考慮する必要のある条件を確認して最終的に AvailForOptimization カラムの値を決定します。

ビューを最適化に使用することをユーザーが許可していますか	ビュー定義は使用に必要な条件をすべて満たしていますか	接続オプションはビューを使用するために必要なオプションと一致していますか	ビューは初期化されていますか	AvailForOptimization 値
Yes	Yes	Yes	Yes	Y

ビューを最適化に使用することをユーザーが許可していますか	ビュー定義は使用に必要な条件をすべて満たしていますか	接続オプションはビューを使用するために必要なオプションと一致していますか	ビューは初期化されていますか	AvailForOptimization 値
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

初期化されたマテリアライズドビューが空であることがあります。これは、基本となるテーブルに、マテリアライズドビュー定義を満たすデータがない場合に発生します。空のビューは、同様にデータを持たない初期化されていないマテリアライズドビューと同じであるとは見なされません。ViewLastRefreshed プロパティの値を使用すると、ビューが初期化されていないのか (NULL)、または基本となるテーブルのデータによって空であるのか (NOT NULL) を区別できます。

パーミッション

DBA 権限。または、DBO が所有するプロシージャーに対する実行パーミッション。

関連する動作

指定したマテリアライズドビューのすべてのメタデータと依存性は、データベースサーバーのキャッシュにロードされます。

参照

- 「マテリアライズドビュー」『SQL Anywhere サーバー SQL の使用法』
- 「高度：マテリアライズドビューの再表示タイプの変更」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューを手動から即時に変更する際の制限」『SQL Anywhere サーバー SQL の使用法』
- 「sa_materialized_view_can_be_immediate システムプロシージャー」 1116 ページ
- 「オプティマイザーによって検討されたマテリアライズドビューの判断」『SQL Anywhere サーバー SQL の使用法』
- 「マテリアライズドビューの制限」『SQL Anywhere サーバー SQL の使用法』
- 「再表示タイプの手動または即時への設定」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、データベース内のすべてのマテリアライズドビューに関する情報を返します。

```
SELECT *
  FROM sa_materialized_view_info();
```

sa_materialized_view_info システムプロシージャーの結果と sa_materialized_view_can_be_immediate システムプロシージャーの結果を組み合わせると、ステー

タス情報だけでなく、ビューを即時ビューに変更できるかどうかも返すことができます。次の文を実行し、この例で検査されるマテリアライズドビューを作成します。

```
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;
```

次の文を実行すると、作成したビューのステータス情報と適格性情報を返します。

```
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
l( CanBeImmediate );;
```

ViewName	Status	ViewLastRefreshed	AvailForOptimization	RefreshType	CanBeImmediate
view0	E	(NULL)	N	I	
view00	E	(NULL)	N	M	
view1	D	(NULL)	N	M	ビュー 'view1' を使用できません。このビューは無効になっています。

ViewName	Status	ViewLastRefreshed	AvailForOptimization	RefreshType	CanBeImmediate
view100	E	2008-02-12 16:47:00.000	Y	M	マテリアライズドビュー view10 を即時に変更できません。すでに初期化されています。マテリアライズド・ビュー view10 を即時に変更できません。Null 入力不可なカラムにユニークなインデックスがありません。マテリアライズドビューを即時に変更できません。COUNT(*) が SELECT リストの一部に必要です。マテリアライズドビューを即時に変更できません。非集合の Null 入力不可のカラムにユニークなインデックスがありません。

結果から、次の内容を確認できます。

●view0 がリフレッシュされていない即時ビューであること。

●view00 がリフレッシュされていない手動ビューであること。

●view1 は無効であること。

●view100 は手動ビューであり、最後にリフレッシュされたのが 2008-02-12 16:47:00.000 であること。

●CanBeImmediate カラムにエラーメッセージがないため、view00 を即時ビューに変更できること。

●CanBeImmediate カラムにリストされている理由のため、view1 と view100 を即時ビューに変更できないこと。

sa_migrate システムプロシージャー

SQL Anywhere データベースにリモートテーブルセットをマイグレートします。

構文

sa_migrate(
base_table_owner,
server_name
[, *table_name*]
[, *owner_name*]
[, *database_name*]
[, *migrate_data*]
[, *drop_proxy_tables*]
[, *migrate_fkeys*]
)

引数

- ***base_table_owner*** SQL Anywhere ターゲットデータベースで、マイグレート後のテーブルを所有するユーザーを指定する VARCHAR(128) パラメーター。GRANT CONNECT 文を使用してこのユーザーを作成します。このパラメーターの値は必須です。

- ***server_name*** リモートデータベースへの接続に使用されるリモートサーバーの名前を指定する VARCHAR(128) パラメーター。CREATE SERVER 文を使用してこのサーバーを作成します。このパラメーターの値は必須です。

- ***table_name*** 単一のテーブルをマイグレートする場合は、この VARCHAR(128) パラメーターを使用してテーブルの名前を指定します。それ以外の場合は、このパラメーターに NULL (デフォルト) を指定します。table_name と owner_name の両方のパラメーターに NULL を指定しないでください。

- ***owner_name*** 1 人の所有者に属するテーブルのみをマイグレートする場合は、この VARCHAR(128) パラメーターを使用して所有者の名前を指定します。それ以外の場合は、このパラメーターに NULL (デフォルト) を入力します。table_name と owner_name の両方のパラメーターに NULL を指定しないでください。

- ***database_name*** リモートデータベースの名前を指定する VARCHAR(128) パラメーター。リモートサーバー上の 1 つのデータベースのみからテーブルをマイグレートする場合は、データベース名を指定します。それ以外の場合は、このパラメーターに NULL (デフォルト) を入力します。

- ***migrate_data*** リモートテーブルのデータをマイグレートするかどうかを指定する任意の BIT パラメーター。このパラメーターは、0 (データをマイグレートしない) または 1 (データ

をマイグレートする) に設定できます。デフォルトでは、データがマイグレートされます。(1)

- ***drop_proxy_tables*** マイグレーションの完了後に、マイグレーションプロセス用に作成されたプロキシテーブルを削除するかどうかを指定する任意の BIT パラメーター。このパラメーターは、0 (プロキシテーブルを削除しない) または 1 (プロキシテーブルを削除する) に設定できます。デフォルトでは、プロキシテーブルが削除されます (1)。

- ***migrate_fkeys*** 外部キーマッピングをマイグレートするかどうかを指定する任意の BIT パラメーター。このパラメーターは、0 (外部キーマッピングをマイグレートしない) または 1 (外部キーマッピングをマイグレートする) に設定できます。デフォルトでは、外部キーマッピングがマイグレートされます (1)。

備考

このプロシージャーを使用して、リモートの Oracle、IBM DB2、Microsoft SQL Server、Adaptive Server Enterprise、または SQL Anywhere のデータベースから SQL Anywhere にテーブルをマイグレートできます。このプロシージャーでは、1 回の操作だけで、指定したサーバーから外部キーマッピングを含めたリモートテーブルセットをマイグレートできます。sa_migrate システムプロシージャーは、次のシステムプロシージャーを呼び出します。

- 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

マイグレーションを柔軟に行う必要がある場合は、sa_migrate の代わりにこれらのシステムプロシージャーを使用できます。たとえば、別のユーザーが所有するテーブルとの外部キー関係を持つテーブルをマイグレートする場合は、sa_migrate を使用すると外部キー関係を保持できません。

テーブルをマイグレートするためには、まず CREATE SERVER 文を使用してリモートデータベースに接続するためのリモートサーバーを作成します。また、CREATE EXTERNLOGIN 文を使用して、リモートデータベースへの外部ログインを作成する必要がある場合もあります。

base_table_owner と *server_name* の各パラメーターを指定するだけで、リモートデータベースのすべてのテーブルを SQL Anywhere データベースにマイグレートできます。ただし、これら 2 つのパラメーターのみを指定した場合、ターゲットの SQL Anywhere データベースでは、マイグレートされたすべてのテーブルが 1 人の所有者に属することになります。リモートデータベースの各テーブルの所有者が異なり、SQL Anywhere データベースでもこれらのテーブルに異なる所有者を指定する場合は、所有者ごとに個別にテーブルをマイグレートし、sa_migrate プロシージャーを呼び出すたびに *base_table_owner* パラメーターと *owner_name* パラメーターを指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

警告
table_name と *owner_name* の両方のパラメーターに NULL を指定しないでください。*table_name* パラメーターと *owner_name* パラメーターの両方に NULL を指定すると、データベース内のシステムテーブルを含む、すべてのテーブルがマイグレートします。また、リモートデータベースで同じ名前で所有者が異なるテーブルは、ターゲットデータベースでは 1 人の所有者に属します。一度にマイグレートするテーブルは、1 人の所有者に関連付けられたテーブルだけにすることをおすすめします。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベース移行」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE EXTERNLOGIN 文」548 ページ
- 「CREATE SERVER 文」618 ページ
- 「GRANT 文」782 ページ
- 「sa_migrate_create_remote_table_list システムプロシージャー」1132 ページ
- 「sa_migrate_create_tables システムプロシージャー」1134 ページ
- 「sa_migrate_data システムプロシージャー」1135 ページ
- 「sa_migrate_create_remote_fks_list システムプロシージャー」1131 ページ
- 「sa_migrate_create_fks システムプロシージャー」1129 ページ
- 「sa_migrate_drop_proxy_tables システムプロシージャー」1136 ページ

例

次の文は、リモートデータベースからユーザー p_chin に属するすべてのテーブルを外部キーマッピングとともにマイグレートし、リモートテーブル内のデータをマイグレートします。また、マイグレーションの完了時にプロキシテーブルを削除します。この例では、マイグレートするすべてのテーブルが、SQL Anywhere ターゲットデータベースの local_user に属します。

```
CALL sa_migrate( 'local_user', 'server_a', NULL, 'p_chin', NULL, 1, 1, 1 );
```

次の文は、ユーザー remote_a に属するテーブルのみをリモートデータベースからマイグレートします。対象となる SQL Anywhere データベースでは、これらのテーブルはユーザーの local_a に所属します。マイグレーション時に作成されるプロキシテーブルは、完了時に削除されません。

```
CALL sa_migrate( 'local_a', 'server_a', NULL, 'remote_a', NULL, 1, 0, 1 );
```

sa_migrate_create_fks システムプロシージャー

dbo.migrate_remote_fks_list テーブルにリストされている各テーブルの外部キーを作成します。

構文

sa_migrate_create_fks(*i_table_owner* **)**

引数

- ***i_table_owner*** SQL Anywhere ターゲットデータベースで、マイグレート後の外部キーを所有するユーザーを指定する VARCHAR(128) パラメーター。異なるユーザーに属するテーブルをマイグレートする場合は、マイグレートするテーブルを所有するユーザーごとに、このプロシージャーを実行します。*i_table_owner* は、GRANT CONNECT 文を使用して作成されます。このパラメーターの値は必須です。

備考

このプロシージャーは、dbo.migrate_remote_fks_list テーブルにリストされている各テーブルの外部キーを作成します。*i_table_owner* 引数で指定したユーザーが、ターゲットデータベース内の外部キーを所有します。

SQL Anywhere ターゲットデータベース内のテーブルの所有者がすべて同じでない場合は、外部キーをマイグレートする必要のあるテーブルを所有しているユーザーごとにこのプロシージャーを実行する必要があります。

注意
このシステムプロシージャーは、その他のマイグレーションシステムプロシージャーと組み合わせて使用されます。このとき、以下の順序で実行する必要があります。

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

または、sa_migrate システムプロシージャーを使用すると、1 回ですべてのテーブルをマイグレートできます。

パーミッション

なし。

関連する動作

なし。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SQL Anywhere へのデータベース移行」『SQL Anywhere サーバー SQL の使用法』
- 「GRANT 文」782 ページ
- 「sa_migrate システムプロシージャー」1127 ページ
- 「sa_migrate_create_remote_table_list システムプロシージャー」1132 ページ
- 「sa_migrate_create_tables システムプロシージャー」1134 ページ
- 「sa_migrate_data システムプロシージャー」1135 ページ
- 「sa_migrate_create_remote_fks_list システムプロシージャー」1131 ページ
- 「sa_migrate_drop_proxy_tables システムプロシージャー」1136 ページ

例

次の文は、dbo.migrate_remote_fks_list テーブルに基づいて外部キーを作成します。SQL Anywhere ローカルデータベースでは、外部キーはユーザー local_a に属します。

```
CALL sa_migrate_create_fks( 'local_a' );
```

sa_migrate_create_remote_fks_list システムプロシージャー

dbo.migrate_remote_fks_list テーブルに移植します。

構文

sa_migrate_create_remote_fks_list(*server_name* **)**

引数

- ***server_name*** リモートデータベースへの接続に使用されるリモートサーバーの名前を指定する VARCHAR(128) パラメーター。リモートサーバーは、CREATE SERVER 文で作成します。このパラメーターの値は必須です。

備考

このプロシージャーは、リモートデータベースからマイグレートできる外部キーのリストを dbo.migrate_remote_fks_list テーブルに移植します。このテーブルから、マイグレートしない外部キーのローを削除できます。

または、sa_migrate システムプロシージャーを使用すると、1 回ですべてのテーブルをマイグレートできます。

このシステムプロシージャーは、他のいくつかのマイグレーションシステムプロシージャーと組み合わせて使用されます。sa_migrate_create_fks システムプロシージャーの備考部分には、マイグレーションプロシージャーのリストと実行順序が記載されています。

パーミッション

なし。

関連する動作

なし。

参照

- 「CREATE SERVER 文」618 ページ
- 「SQL Anywhere へのデータベース移行」『SQL Anywhere サーバー SQL の使用法』
- 「sa_migrate システムプロシージャー」1127 ページ
- 「sa_migrate_create_remote_table_list システムプロシージャー」1132 ページ
- 「sa_migrate_create_tables システムプロシージャー」1134 ページ
- 「sa_migrate_data システムプロシージャー」1135 ページ
- 「sa_migrate_create_fks システムプロシージャー」1129 ページ
- 「sa_migrate_drop_proxy_tables システムプロシージャー」1136 ページ

例

次の文は、リモートデータベースにある外部キーのリストを作成します。

```
CALL sa_migrate_create_remote_fks_list( 'server_a' );
```

sa_migrate_create_remote_table_list システムプロシージャー

dbo.migrate_remote_table_list テーブルに移植します。

構文

sa_migrate_create_remote_table_list(
i_server_name
[, *i_table_name*
[, *i_owner_name*
[, *i_database_name*]]]
)

引数

- ***i_server_name*** リモートデータベースへの接続に使用されるリモートサーバーの名前を指定する VARCHAR(128) パラメーター。リモートサーバーは、CREATE SERVER 文で作成します。このパラメーターの値は必須です。

- ***i_table_name*** マイグレートするテーブルの名前を指定する任意の VARCHAR(128) パラメーター。すべてのテーブルをマイグレートする場合は NULL を指定します。デフォルト値は NULL です。*i_table_name* と *i_owner_name* の両方のパラメーターに NULL を指定しないでください。

- ***i_owner_name*** リモートデータベースで、マイグレートするテーブルを所有するユーザーを指定する任意の VARCHAR(128) パラメーター。すべてのテーブルをマイグレートする場合は NULL を指定します。デフォルト値は NULL です。*i_table_name* と *i_owner_name* の両方のパラメーターに NULL を指定しないでください。

- ***i_database_name*** テーブルのマイグレート元のリモートデータベースの名前を指定する任意の VARCHAR(128) パラメーター。このパラメーターのデフォルトは NULL です。Adaptive Server Enterprise と Microsoft SQL Server データベースからテーブルをマイグレートする場合は、データベース名を指定する必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このプロシージャーは、リモートデータベースからマイグレートできるテーブルのリストを dbo.migrate_remote_table_list テーブルに移植します。このテーブルから、マイグレートしないリモートテーブルのローを削除できます。

SQL Anywhere ターゲットデータベースで、マイグレート後のテーブルをすべて同じ所有者に属さないようにする場合は、マイグレートするテーブルを所有するユーザーごとにこのプロシージャーを実行します。

または、sa_migrate システムプロシージャーを使用すると、1 回ですべてのテーブルをマイグレートできます。

警告
i_table_name と *i_owner_name* の両方のパラメーターに NULL を指定しないでください。*i_table_name* パラメーターと *i_owner_name* パラメーターの両方に NULL を指定すると、システムテーブルを含む、データベース内のすべてのテーブルがマイグレートされます。また、リモートデータベースで同じ名前で所有者が異なるテーブルは、ターゲットデータベースでは 1 人の所有者に属します。一度にマイグレートするテーブルは、1 人の所有者に関連付けられたテーブルだけにすることをおすすめします。

このシステムプロシージャーは、他のいくつかのマイグレーションシステムプロシージャーと組み合わせて使用されます。sa_migrate_create_fks システムプロシージャーの備考部分には、マイグレーションプロシージャーのリストと実行順序が記載されています。

パーミッション

なし。

関連する動作

なし。

参照

- 「CREATE SERVER 文」618 ページ
- 「SQL Anywhere へのデータベース移行」『SQL Anywhere サーバー SQL の使用法』
- 「sa_migrate システムプロシージャー」1127 ページ
- 「sa_migrate_create_tables システムプロシージャー」1134 ページ
- 「sa_migrate_data システムプロシージャー」1135 ページ
- 「sa_migrate_create_remote_fks_list システムプロシージャー」1131 ページ
- 「sa_migrate_create_fks システムプロシージャー」1129 ページ
- 「sa_migrate_drop_proxy_tables システムプロシージャー」1136 ページ

例

次の文は、リモートデータベース上のユーザー remote_a に属するテーブルのリストを作成します。

```
CALL sa_migrate_create_remote_table_list( 'server_a', NULL, 'remote_a', NULL );
```

sa_migrate_create_tables システムプロシージャー

dbo.migrate_remote_table_list テーブルにリストされている各テーブルのプロキシテーブルとベーステーブルを作成します。

構文

sa_migrate_create_tables(*i_table_owner* **)**

引数

- ***i_table_owner*** SQL Anywhere ターゲットデータベースで、マイグレート後のテーブルを所有するユーザーを指定する VARCHAR(128) パラメーター。このユーザーは、GRANT CONNECT 文を使用して作成します。このパラメーターの値は必須です。

備考

このプロシージャーは、dbo.migrate_remote_table_list テーブル (sa_migrate_create_remote_table_list プロシージャーを使用して作成したテーブル) にリストされている各テーブルに対して、ベーステーブルとプロキシテーブルを作成します。これらのプロキシテーブルとベーステーブルは、*i_table_owner* 引数で指定したユーザーが所有します。このプロシージャーは、リモートデータベースのリモートテーブルと同じプライマリキーインデックスとその他のインデックスも、新しいテーブルに対して作成します。

SQL Anywhere ターゲットデータベースで、マイグレート後のすべてのテーブルの所有者を同じにしたくない場合は、マイグレートするテーブルを所有するユーザーごとに、sa_migrate_create_remote_table_list プロシージャーと sa_migrate_create_tables プロシージャーを実行します。

または、sa_migrate システムプロシージャーを使用すると、1 回ですべてのテーブルをマイグレートできます。

このシステムプロシージャーは、他のいくつかのマイグレーションシステムプロシージャーと組み合わせて使用されます。sa_migrate_create_fks システムプロシージャーの備考部分には、マイグレーションプロシージャーのリストと実行順序が記載されています。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベース移行」『SQL Anywhere サーバー SQL の使用法』
- 「sa_migrate システムプロシージャー」1127 ページ
- 「sa_migrate_create_remote_table_list システムプロシージャー」1132 ページ
- 「sa_migrate_data システムプロシージャー」1135 ページ
- 「sa_migrate_create_remote_fks_list システムプロシージャー」1131 ページ
- 「sa_migrate_create_fks システムプロシージャー」1129 ページ
- 「sa_migrate_drop_proxy_tables システムプロシージャー」1136 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、SQL Anywhere ターゲットデータベース上にベーステーブルとプロキシテーブルを作成します。これらのテーブルは、ユーザー local_a に属します。

```
CALL sa_migrate_create_tables( 'local_a' );
```

sa_migrate_data システムプロシージャー

リモートデータベーステーブルから SQL Anywhere ターゲットデータベースにデータをマイグレートします。

構文

sa_migrate_data(*i_table_owner* **)**

引数

- ***i_table_owner*** SQL Anywhere ターゲットデータベースで、マイグレート後のテーブルを所有するユーザーを指定する VARCHAR(128) パラメーター。このユーザーは、GRANT CONNECT 文を使用して作成します。このパラメーターの値は必須です。

備考

このプロシージャーは、リモートデータベースから SQL Anywhere データベースに、*i_table_owner* 引数で指定したユーザーに属するテーブルのデータをマイグレートします。

SQL Anywhere ターゲットデータベース上のテーブルの所有者がすべて同じでない場合は、データをマイグレートするテーブルを所有しているユーザーごとに、このプロシージャーを実行する必要があります。

または、sa_migrate システムプロシージャーを使用すると、1 回ですべてのテーブルをマイグレートできます。

このシステムプロシージャーは、他のいくつかのマイグレーションシステムプロシージャーと組み合わせて使用されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベース移行」『SQL Anywhere サーバー SQL の使用法』
- 「sa_migrate システムプロシージャー」1127 ページ
- 「sa_migrate_create_remote_table_list システムプロシージャー」1132 ページ
- 「sa_migrate_create_tables システムプロシージャー」1134 ページ
- 「sa_migrate_create_remote_fks_list システムプロシージャー」1131 ページ
- 「sa_migrate_create_fks システムプロシージャー」1129 ページ
- 「sa_migrate_drop_proxy_tables システムプロシージャー」1136 ページ

例

次の文は、ユーザー local_a に属するテーブルのデータを SQL Anywhere ターゲットデータベースにマイグレートします。

```
CALL sa_migrate_data( 'local_a' );
```

sa_migrate_drop_proxy_tables システムプロシージャー

マイグレーションの目的で作成されたプロキシテーブルを削除します。

構文

sa_migrate_drop_proxy_tables(*i_table_owner* **)**

引数

- ***i_table_owner*** SQL Anywhere ターゲットデータベースで、プロキシテーブルを所有するユーザーを指定する VARCHAR(128) パラメーター。このユーザーは、GRANT CONNECT 文を使用して作成します。このパラメーターの値は必須です。

備考

このプロシージャーは、マイグレーションのために作成されたプロキシテーブルを削除します。これらのプロキシテーブルを所有するユーザーを、*i_table_owner* 引数で指定します。

SQL Anywhere ターゲットデータベースで、マイグレート後のテーブルをすべて同じユーザーが所有しない場合、ユーザーごとにこのプロシージャーを呼び出して、すべてのプロキシテーブルを削除します。

または、sa_migrate システムプロシージャーを使用すると、1 回ですべてのテーブルをマイグレートできます。

このシステムプロシージャーは、他のいくつかのマイグレーションシステムプロシージャーと組み合わせて使用されます。sa_migrate_create_fks システムプロシージャーの備考部分には、マイグレーションプロシージャーのリストと実行順序が記載されています。

パーミッション

なし。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベース移行」『SQL Anywhere サーバー SQL の使用法』
- 「sa_migrate システムプロシージャー」1127 ページ
- 「sa_migrate_create_remote_table_list システムプロシージャー」1132 ページ
- 「sa_migrate_create_tables システムプロシージャー」1134 ページ
- 「sa_migrate_data システムプロシージャー」1135 ページ
- 「sa_migrate_create_remote_fks_list システムプロシージャー」1131 ページ
- 「sa_migrate_create_fks システムプロシージャー」1129 ページ

例

次の文は、SQL Anywhere ターゲットデータベースで、ユーザー local_a に属するプロキシテーブルを削除します。

```
CALL sa_migrate_drop_proxy_tables( 'local_a' );
```

sa_mirror_server_status システムプロシージャー

現在のサーバーと現在のサーバーから直接または間接的にログページを受信するすべてのサーバーの接続ステータスを返します。プライマリサーバーでは、接続されているすべてのサーバーのステータスを返します。

構文

sa_mirror_server_status( )

結果セット

カラム名	データ型	説明
server_name	CHAR(128)	サーバーの名前。
state	CHAR(20)	サーバーの接続ステータス。値は次のいずれかです。 ● connected ● disconnected
last_updated	TIMESTAMP WITH TIME ZONE	サーバーステータスが最後に更新された時刻。
load_current	DOUBLE	データベースサーバーで現在実行されている作業量。

カラム名	データ型	説明
load_last_1_min	DOUBLE	データベースサーバーで過去 1 分間で実行された作業量。
load_last_5_mins	DOUBLE	データベースサーバーで過去 5 分間で実行された作業量。
load_last_10_mins	DOUBLE	データベースサーバーで過去 10 分間で実行された作業量。
num_connections	UNSIGNED INT	データベースサーバーとの接続の数。
num_processors	UNSIGNED INT	データベースサーバープロセッサーの数。
log_written	UNSIGNED BIGINT	サーバーから受信した最後の更新に基づいてディスクに書き込まれた最新のトランザクションログの位置。
log_applied	UNSIGNED BIGINT	サーバーから受信した最後の更新に基づいて適用されたトランザクションログ内の最後の操作。この値は、CurrentRedoPos プロパティの値と同じです。

備考

各サーバーでは、サーバー自身とそのコピーノードのステータスが 5 秒ごとに更新されます。ミラーサーバーの場合、ミラーサーバーからログページを受信するすべてのコピーノードのステータスが返りますが、プライマリサーバーのステータスは返りません。プレフィクス 「load」 が付いたカラムは、データベースサーバー上の負荷の計算値を表します。返される値は、データベースサーバーの負荷を表し、他のプロセスからの負荷ではありません。負荷値が高い場合、データベースサーバーで実行される作業量が多いことを示します。

NodeType 接続パラメーターを指定すると、データベースサーバーでは負荷情報に基づいて接続をリダイレクトします。データベースサーバーは負荷が最も低いミラーサーバーを選択します。すべてのサーバーの負荷が同じ場合は、接続数が最も少ないサーバーが使用されます。

注意
クラウドで実行されているデータベースサーバーの場合、NodeType 接続パラメーターによる接続のリダイレクトに負荷情報は使用されません。

パーミッション

なし。

関連する動作

なし。

参照

- CurrentRedoPos データベースプロパティ『SQL Anywhere サーバー データベース管理』
- 「NodeType (NODE) 接続パラメーター」『SQL Anywhere サーバー データベース管理』
- 「データベースミラーリング」『SQL Anywhere サーバー データベース管理』
- 「CREATE MIRROR SERVER 文」579 ページ
- 「ALTER MIRROR SERVER 文」430 ページ
- 「DROP MIRROR SERVER 文」719 ページ

sa_nchar_terms システムプロシージャー

NCHAR 文字列を単語に分割し、各単語とその位置を 1 つのローとして返します。

構文

sa_nchar_terms('*text*' [, '*config_name*' [, '*owner*']]
)

引数

- ***text*** 解析する NCHAR 文字列。
- ***config_name*** 文字列を処理するときに適用されるテキスト設定オブジェクト。デフォルトは 'default_nchar' です。
- ***owner*** 指定されたテキスト設定オブジェクトの所有者。デフォルト値は NULL です。また、所有者が指定されていない場合、または NULL が指定された場合は、現在のユーザーが想定されます。

備考

このシステムプロシージャーを使用すると、テキスト設定オブジェクトの設定が適用されたときに文字列がどのように解釈されるかを確認できます。インデックス処理で、またはクエリ文字列から、どの単語が削除されるかを確かめる場合に便利です。

パーミッション

なし。

関連する動作

なし。

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「sa_char_terms システムプロシージャー」1043 ページ

例

次の文は、デフォルトの NCHAR テキスト設定オブジェクト default_nchar を使用して、NCHAR 文字列 "It's a work-at-home day!" 内の単語を返します。

```
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

sa_performance_diagnostics システムプロシージャー

データベースサーバーが要求のタイミングの記録を有効にしたとき、すべての接続について要求のタイミング情報の概要を返します。

構文

sa_performance_diagnostics( )

結果セット

カラム名	データ型	説明
Number	INT	接続の ID 番号を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
Name	VARCHAR(255)	現在の接続の名前を返します。 ConnectionName (CON) 接続パラメーターを使用して接続名を指定できます。「ConnectionName (CON) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。 データベースサーバーによって作成されるテンポラリ接続には次の名前が使用されます。 ● 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 「テンポラリ接続」『SQL Anywhere サーバー データ

カラム名	データ型	説明
		ベース管理』を参照してください。
Userid	VARCHAR(255)	接続のユーザー ID を返します。
DBNumber	INT	データベースの ID 番号を返します。
LoginTime	TIMESTAMP	接続が確立された日付と時刻を返します。
TransactionStartTime	TIMESTAMP	COMMIT または ROLLBACK の後にデータベースが最初に変更された時刻を含む文字列を返します。最後の COMMIT または ROLLBAK 以降にデータベースが変更されていない場合は、空の文字列を返します。
LastReqTime	TIMESTAMP	指定された接続において最後の要求が開始された時刻を返します。このプロパティは、イベントなどの内部接続の場合は空の文字列を返すことがあります。
ReqType	VARCHAR(255)	最後の要求のタイプを返します。接続が接続プールによってキャッシュされる場合、その ReqType 値は CONNECT_POOL_CACHE になります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
ReqStatus	VARCHAR(255)	要求のステータスを返します。値は次のいずれかです。 ● **Idle** この接続では現在、要求を処理していない。 ● **Unscheduled*** この接続では他の処理が実行されており、使用可能なデータベースサーバーワーカーを待っている。 ● **BlockedIO*** この接続はブロックされ、I/O 処理の完了を待っている。 ● **BlockedContention*** この接続はブロックされ、共有データベースサーバーデータ構造体へのアクセスを待っている。 ● **BlockedLock** この接続はブロックされ、オブジェクトのロックの解放を待っている。 ● **Executing** この接続では現在、要求を実行している。 アスタリスク (*) が付いている値が返されるのは、-zt サーバーオプションによってデータベースサーバーで要求タイミング情報のロギングがオンになっている場合に限られます。要求タイミング情報のロギングが実行されていない場合は (デフォルト)、値は Executing として返されます。 「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバーデータベース管理』を参照してください。

カラム名	データ型	説明
ReqTimeUnscheduled	DOUBLE	未スケジュール時間を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqTimeActive	DOUBLE	要求の処理に要した時間を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqTimeBlockIO	DOUBLE	I/O 処理が完了するまでの待ち時間を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqTimeBlockLock	DOUBLE	ロックが解放されるまでの待ち時間を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqTimeBlockContention	DOUBLE	アトミックアクセスを取得するまでの待ち時間を返します。RequestTiming サーバープロパティが Off に設定されている場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
ReqCountUnscheduled	INT	接続がスケジューリングを待った回数を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqCountActive	INT	処理が完了した要求の数を返します。RequestTiming サーバープロパティが Off に設定されている場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqCountBlockIO	INT	接続が I/O 処理の完了を待った回数を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqCountBlockLock	INT	接続がロックの解放を待った回数を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。
ReqCountBlockContention	INT	接続がアトミックアクセスを待った回数を返します。-zt オプションが指定されていない場合は、NULL を返します。「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

カラム名	データ型	説明
LastIdle	INT	要求間のチック数を返します。
BlockedOn	INT	現在の接続がブロックされていない場合は 0 を返し、ブロックされている場合はロック競合によってブロックされた接続の接続番号を返します。
UncommitOp	INT	コミットされていない操作の数を返します。
CurrentProcedure	VARCHAR(255)	接続で現在実行されているプロシージャーの名前を返します。接続でネストされたプロシージャーコールが実行されている場合は、現在のプロシージャーの名前を返します。実行されているプロシージャーがない場合は、空の文字列を返します。
EventName	VARCHAR(255)	接続でイベントハンドラーが実行されている場合、関連付けられているイベントの名前を返します。それ以外の場合は、空の文字列が返されます。
CurrentLineNumber	INT	接続で実行されているプロシージャーまたは複合文の現在の行番号を返します。実行されているプロシージャーの名前は CurrentProcedure プロパティで取得できます。現在の行がクライアント側からの複合文の一部である場合は、空の文字列を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
LastStatement	LONG VARCHAR	現在の接続で最後に準備された SQL 文を返します。 LastStatement の値は、文が準備されると同時に設定され、文が削除されると同時にクリアされます。各接続につき 1 つの文の文字列のみが記憶されます。 ある接続について sa_conn_activity が空でない値を返した場合、その接続で現在実行されている文である可能性が高くなります。その文が完了している場合は、文がすでに削除され、このプロパティの値がクリアされている可能性があります。アプリケーションが複数の文を準備し、それらの文のステートメントハンドルを保持している場合、LastStatement が返す値は接続で現在実行されている処理を表しません。 クライアントでの文のキャッシュが有効であり、キャッシュされた文が再使用されているとき、このプロパティは空の文字列を返します。

カラム名	データ型	説明
LastPlanText	LONG VARCHAR	接続で最後に実行されたクエリの長いテキストプランを返します。最後のプランを記憶するかどうかは、sa_server_option システムプロシージャーの RememberLastPlan オプションを設定するか、-zp サーバーオプションを使用することで制御します。「-zp dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』を参照してください。
AppInfo	LONG VARCHAR	接続を確立したクライアントに関する情報を返します。HTTP 接続では、ブラウザーの情報が含まれています。jConnect または Sybase Open Client の古いバージョンを使った接続については、情報は不完全の場合があります。 API 値は、DBLIB、ODBC、OLEDB、ADO.NET、iAnywhereJDBC、PHP、PerlDBD、DBEXPRESS のいずれかです。 その他のタイプの接続について返される値の詳細については、「AppInfo (APP) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。
LockCount	INT	接続で保持されているロックの数を返します。
SnapshotCount	INT	接続に関連付けられているスナップショットの数を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

sa_performance_diagnostics システムプロシージャーは、要求のタイミングプロパティで構成される結果セットと統計情報 (統計情報を収集するようにサーバーに指示されている場合) を返します。要求のタイミング情報の記録は、sa_performance_diagnostics の呼び出し前にデータベースサーバーでオンにする必要があります。オンにするには、データベースサーバーの起動時に -zt オプションを指定するか、次の文を実行します。

```
CALL sa_server_option( 'RequestTiming','ON' );
```

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』
- 「sa_performance_statistics システムプロシージャー」1149 ページ
- 「sa_server_option システムプロシージャー」1170 ページ

例

次のクエリを実行すると、データベースサーバー要求が完了するまでの待ち時間が長い接続が識別されます。

```
SELECT  Number, Name,
    CAST( DATEDIFF( second, LoginTime, CURRENT TIMESTAMP ) AS DOUBLE ) AS T,
    ReqTimeActive / T AS PercentActive
FROM  dbo.sa_performance_diagnostics()
WHERE PercentActive > 10.0
ORDER BY PercentActive DESC;
```

現在実行中で、60 秒よりも長く実行されているすべての要求を検索します。

```
SELECT  Number, Name,
    CAST( DATEDIFF( second, LastReqTime, CURRENT TIMESTAMP ) AS DOUBLE ) AS ReqTime
FROM  dbo.sa_performance_diagnostics()
WHERE ReqStatus <> 'IDLE' AND ReqTime > 60.0
ORDER BY ReqTime DESC;
```

sa_performance_statistics システムプロシージャー

データベースサーバーが要求のタイミングの記録を有効にしたとき、すべての接続についてメモリ診断の統計情報の概要を返します。

構文

sa_performance_statistics( )

結果セット

カラム名	データ型	説明
DBNumber	INT	データベースの ID 番号を返します。
ConnNumber	INT	接続の ID 番号を返します。 Type が Server の場合は NULL が返されます。
PropNum	INT	接続プロパティの番号が返されます。
PropName	VARCHAR(255)	接続プロパティの名前が返されます。
Value	INT	接続プロパティの値が返されます。

備考

sa_performance_statistics システムプロシージャーは、メモリ診断の統計情報で構成される結果セット (統計情報を収集するようにサーバーに指示されている場合) を返します。メモリ診断の統計情報の記録は、sa_performance_statistics の呼び出し前にデータベースサーバーでオンにする必要があります。オンにするには、データベースサーバーの起動時に -zt オプションを指定するか、次の文を実行します。

```
CALL sa_server_option( 'RequestTiming','ON' );
```

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』
- 「sa_performance_diagnostics システムプロシージャー」1140 ページ
- 「sa_server_option システムプロシージャー」1170 ページ

例

次の例は、すべてのパフォーマンス統計をテキストファイル *dump_stats.txt* にアンロードします。

```
UNLOAD
  SELECT CURRENT TIMESTAMP, *
  FROM sa_performance_statistics()
  TO 'dump_stats.txt'
  APPEND ON;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

sa_post_login_procedure システムプロシージャー

ユーザーのパスワードの有効期限切れが近づいているかどうかを判断します。

構文

sa_post_login_procedure( )

引数

なし。

結果セット

sa_post_login_procedure システムプロシージャーは、次の結果を返します。

カラム名	データ型	説明
message_text	VARCHAR(255)	message_action が 1 の場合、message_text は表示するメッセージを返します。 message_action が 0 の場合、message_text は NULL です。
message_action	INTEGER	パスワードの有効期限切れが近づいているかどうか (1= はい、0= いいえ)。

備考

sa_post_login_procedure システムプロシージャーは、post_login_procedure データベースオプションのデフォルト設定です。

sa_post_login_procedure は、ユーザーの password_life_time と password_grace_time の各ログインポリシーオプション値および現在の日付と時刻を使用して、ユーザーのパスワードの有効期限切れが近づいているかどうかを判断します。有効期限切れに近い場合は、ユーザーに表示するメッセージが結果セットに返されます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「post_login_procedure オプション」『SQL Anywhere サーバー データベース管理』
- 「ログインポリシー」『SQL Anywhere サーバー データベース管理』

sa_procedure_profile システムプロシージャー

データベースで実行されたプロシージャー、関数、イベント、またはトリガー内の各行について、実行時間に関する情報をレポートします。

構文

sa_procedure_profile(
[*filename*
[, *save_to_file*]]
)

引数

- ***filename*** 任意の LONG VARCHAR パラメーターを使用して、プロファイル情報を保存するファイル、またはロードするファイルを指定します。プロファイル情報の保存とロードの詳細については、以下の備考部分を参照してください。

- ***save_to_file*** プロファイル情報をファイルに保存するか、前に保存したファイルからロードするかを指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
object_type	CHAR(1)	オブジェクトのタイプ。使用できるオブジェクトタイプのリストについては、以下の備考部分を参照してください。
object_name	CHAR(128)	ストアドプロシージャー、関数、イベント、またはトリガーの名前。object_type が C または D の場合、これはシステムトリガーが定義された外部キーの名前です。
owner_name	CHAR(128)	オブジェクトの所有者。
table_name	CHAR(128)	トリガーに対応するテーブル (他のオブジェクトタイプの場合、値は NULL)。
line_num	UNSIGNED INTEGER	プロシージャー内の行番号。
executions	UNSIGNED INTEGER	行の実行回数。
millisecs	UNSIGNED INTEGER	行の実行時間 (ミリ秒単位)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
percentage	DOUBLE	全体的な実行時間に対する特定の行の実行時間の割合。
foreign_owner	CHAR(128)	システムトリガーの外部テーブルを所有するデータベースユーザー。
foreign_table	CHAR(128)	システムトリガーの外部テーブルの名前。

備考

このプロシージャーは、Sybase Central の **[プロファイル]** タブと同じ情報を提供します。

このプロシージャーは、次の用途で使用できます。

- **詳細なプロシージャーのプロファイル情報を返す**　この場合、引数指定することなく、プロシージャーを呼び出すだけです。
- **詳細なプロシージャーのプロファイル情報をファイルに保存**　この場合、*filename* 引数を含め、*save_to_file* 引数に 1 を指定します。
- **詳細なプロシージャーのプロファイル情報を前に保存したファイルからロード**　この場合、*filename* 引数を含め、*save_to_file* 引数に 0 を指定します (デフォルトが 0 のため、そのまま何も指定しなくてもかまいません)。この方法でプロシージャーを使用できるのは、ロードするファイルが、プロシージャーを実行するデータベースと同じデータベースで作成されたものである場合です。それ以外の場合、結果は使用できない可能性があります。

結果セットには、プロシージャー、トリガー、関数、イベント内の個々の行に関する実行回数の情報と、行が使用するプロシージャーの合計の実行回数の割合が含まれるため、このプロファイル情報を使用して、パフォーマンスを低下させる可能性がある遅いプロシージャーを微調整できます。

プロファイリングを有効にしてから、データベースのプロファイルを作成します。

結果セットの object_type カラムは、次のようになります。

- **P**　ストアドプロシージャー
- **F**　関数
- **E**　イベント
- **T**　トリガー
- **C**　ON UPDATE システムトリガー
- **D**　ON DELETE システムトリガー

各実行について、行ごとの詳細ではなく、概要情報が必要な場合、sa_procedure_profile_summary プロシージャーを使用します。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_server_option システムプロシージャー」1170 ページ
- 「sa_procedure_profile_summary システムプロシージャー」1154 ページ
- 「プロシージャープロファイリングの有効化」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、データベースで実行されたすべてのプロシージャー、関数、イベント、またはトリガーの各行について実行時間を返します。

```
CALL sa_procedure_profile( );
```

次の文は、上記の例と同じ詳細なプロシージャーのプロファイル情報を返し、この情報をファイル *detailedinfo.txt* に保存します。

```
CALL sa_procedure_profile( "detailedinfo.txt", 1 );
```

次の文はいずれも、ファイル *detailedinfoOLD.txt* から詳細なプロシージャーのプロファイル情報をロードするときに使用できます。

```
CALL sa_procedure_profile( "detailedinfoOLD.txt", 0 );
CALL sa_procedure_profile( "detailedinfoOLD.txt" );
```

sa_procedure_profile_summary システムプロシージャー

データベースで実行されたすべてのプロシージャー、ファンクション、イベント、またはトリガーの実行時間に関する概要情報をレポートします。このプロシージャーは、Sybase Central の **[プロファイル]** タブと同じオブジェクトの情報を提供します。

構文

sa_procedure_profile_summary(
  [*filename*
  [, *save_to_file*]]
)

引数

- ***filename*** プロファイル情報を保存するファイル、またはロードするファイルを指定する任意の LONG VARCHAR パラメーター。プロファイル情報の保存とロードの詳細については、以下の備考部分を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- ***save_to_file*** 概要情報をファイルに保存するか、前に保存したファイルからロードするかを指定する任意の INTEGER パラメーター。

結果セット

カラム名	データ型	説明
object_type	CHAR(1)	オブジェクトのタイプ。使用できるオブジェクトタイプのリストについては、以下の備考部分を参照してください。
object_name	CHAR(128)	ストアドプロシージャー、ファンクション、イベント、またはトリガーの名前。
owner_name	CHAR(128)	オブジェクトの所有者。
table_name	CHAR(128)	トリガーに対応するテーブル (他のオブジェクトタイプの場合、値は NULL)。
executions	UNSIGNED INTEGER	各プロシージャーの実行回数。
millisecs	UNSIGNED INTEGER	プロシージャーの実行時間 (ミリ秒単位)。
foreign_owner	CHAR(128)	システムトリガーの外部テーブルを所有するデータベースユーザー。
foreign_table	CHAR(128)	システムトリガーの外部テーブルの名前。

備考

このプロシージャーは、次の用途で使用できます。

- **現在の概要情報を返します。** この場合、引数指定することなく、プロシージャーを呼び出すだけです。

- **現在の概要情報をファイルに保存** この場合、*filename* 引数を含め、*save_to_file* 引数に 1 を指定します。

- **保存された概要情報をファイルからロード** この場合、*filename* 引数を含め、*save_to_file* 引数に 0 を指定します (デフォルトが 0 のため、そのまま何も指定しなくてもかまいません)。この方法でプロシージャーを使用できるのは、ロードするファイルが、プロシージャーを実行するデータベースと同じデータベースで作成されたものである場合です。それ以外の場合、結果は使用できない可能性があります。

このプロシージャーは、ストアドプロシージャー、ファンクション、イベント、トリガーの使用頻度や効率に関する情報を返すため、この情報を使用して、時間がかかるプロシージャーを微調整してデータベースのパフォーマンスを改善できます。

プロファイリングを有効にしてから、データベースのプロファイルを作成します。

結果セットの object_type カラムは、次のようになります。

- **P** ストアドプロシージャー
- **F** ファンクション
- **E** イベント
- **T** トリガー
- **S** システムトリガー
- **C** ON UPDATE システムトリガー
- **D** ON DELETE システムトリガー

各実行について、概要情報ではなく、行ごとの詳細が必要な場合、sa_procedure_profile プロシージャーを使用します。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「プロシージャープロファイリングの有効化」『SQL Anywhere サーバー SQL の使用法』
- 「sa_server_option システムプロシージャー」1170 ページ
- 「sa_procedure_profile システムプロシージャー」1152 ページ

例

次の文は、データベースで実行されたすべてのプロシージャー、ファンクション、イベント、またはトリガーについて実行時間を返します。

```
CALL sa_procedure_profile_summary( );
```

次の文は、前の例と同じ概要情報を返し、この情報をファイル *summaryinfo.txt* に保存します。

```
CALL sa_procedure_profile_summary( "summaryinfo.txt", 1 );
```

次の文はいずれも、ファイル *summaryinfoOLD.txt* に保存されている概要情報をロードするときに使用できます。

```
CALL sa_procedure_profile_summary( "summaryinfoOLD".txt, 0 );
```

```
CALL sa_procedure_profile_summary( "summaryinfoOLD.txt" );
```

sa_recompile_views システムプロシージャー

カタログに格納された、カラム定義を持たないビュー定義を見つけ、カラム定義を作成します。

構文

sa_recompile_views([*ignore_errors*])

引数

- ***ignore_errors*** 再コンパイル中に発生したエラーを返すかどうかを指定する任意の INTEGER パラメーター。0 と指定すると、カラム定義が失敗したビューごとにエラーが返されます。1 または 0 以外の値を指定すると、エラーは返されません。値を指定しない場合のデフォルト値は 0 です。

備考

このプロシージャーは、カラム定義を持たないビューをカタログで見つけ、RECOMPILE 句を指定した ALTER VIEW 文を実行してカラム定義を作成します。プロシージャーは、コンパイルを必要とするものがなくなるまで、または残りのカラム定義を作成できなくなるまで、カラム定義を持たない各ビューに対してこの処理を実行します。プロシージャーでビューを再コンパイルできない場合は、エラーがレポートされます。このプロシージャーに 0 以外のパラメーターを指定することによって、エラーを抑制できます。

警告
sa_recompile_views システムプロシージャーは、*reload.sql* スクリプト内からのみ呼び出してください。このプロシージャーはアンロードユーティリティ (dbunload) によって使用されます。明示的には使用しないでください。

sa_recompile_views システムプロシージャーは、マテリアライズドビューまたは DISABLED と印が付いているビューを再コンパイルしません。

パーミッション

DBA 権限。

関連する動作

VALID ステータスのない通常の各ビューについては、ALTER VIEW owner.viewname ENABLE 文が実行され、オートコミットが行われます。

参照

- 「通常のビューのステータス」『SQL Anywhere サーバー SQL の使用法』
- 「force_view_creation オプション」『SQL Anywhere サーバー データベース管理』
- 「ALTER VIEW 文」484 ページ

sa_refresh_materialized_views システムプロシージャー

初期化されていないステータスであるすべてのマテリアライズドビューを初期化します。

構文

sa_refresh_materialized_views([*ignore_errors*])

引数

- ***ignore_errors*** 再コンパイル中に発生したエラーを返すかどうかを指定する任意の INTEGER パラメーター。0 と指定すると、カラム定義が失敗したビューごとにエラーが返されます。1 または 0 以外の値を指定すると、エラーは返されません。値を指定しない場合のデフォルト値は 0 です。

備考

作成したばかりのため、再有効化したばかりのため、初期化や更新の最後の実行がエラーにより失敗したために、マテリアライズドビューが未初期化ステータスになることがあります。sa_refresh_materialized_views システムプロシージャーは、このようなすべてのマテリアライズドビューについてデータベースをスキャンし、初期化を試みます。プロシージャーでマテリアライズドビューの初期化時にエラーが発生した場合、残りの未初期化ビューの処理が続けられます。

また、REFRESH MATERIALIZED VIEW 文を使用して、マテリアライズドビューを初期化することもできます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「REFRESH MATERIALIZED VIEW 文」867 ページ
- 「手動マテリアライズドビューのリフレッシュ」『SQL Anywhere サーバー SQL の使用法』

sa_refresh_text_indexes システムプロシージャー

MANUAL REFRESH または AUTO REFRESH と定義されているすべてのテキストインデックスを再表示します。

構文

sa_refresh_text_indexes( )

備考

sa_refresh_text_indexes システムプロシージャーは、MANUAL REFRESH または AUTO REFRESH と定義されているすべてのテキストインデックスを再表示します。IMMEDIATE REFRESH (デ

フォルト) と定義されているテキストインデックスは再表示されません。これは、このようなインデックスの変更は、基本となるテーブルでデータが変更されたときに行われるためです。

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキスト設定オブジェクトの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ
- 「TRUNCATE 文」961 ページ
- 「SYSTEXTIDX システムビュー」1322 ページ
- 「sa_text_index_stats システムプロシージャー」1201 ページ
- 「sa_text_index_vocab システムプロシージャー」1202 ページ

例

次の文は、データベースに含まれるすべての MANUAL REFRESH と AUTO REFRESH のテキストインデックスを再表示します。

```
CALL sa_refresh_text_indexes( );
```

sa_remove_tracing_data システムプロシージャー

診断トレーシングテーブルから、指定したロギング (トレーシング) セッション ID に関連するすべてのレコードを永久的に削除します。

構文

sa_remove_tracing_data(*log_session_id* **)**

引数

- ***log_session_id*** データを削除するロギングセッションの ID を指定する INTEGER パラメーター。

備考

指定した *log_session_id* にレコードがない場合、プロシージャーでは何も行われません。このプロシージャーには戻り値がありません。

パーミッション

DBA 権限。

関連する動作

指定した *log_session_id* にレコードが見つからなかった場合でも、完了時にコミットが発生します。

参照

- 「診断トレーシングテーブル」1008 ページ

sa_report_deadlocks システムプロシージャー

データベースサーバーによって作成された内部バッファーから、デッドロックに関する情報を取り出します。

構文

sa_report_deadlocks( )

結果セット

カラム名	データ型	説明
snapshotId	BIGINT	デッドロックインスタンス (特定のデッドロックに関するすべてのローが同じ ID を持ちます)。
snapshotAt	TIMESTAMP	デッドロックが発生した時刻。
waiter	INT	待機している接続の接続ハンドル。
who	VARCHAR(128)	待機している接続に関連付けられているユーザー ID。
what	LONG VARCHAR	待機している接続によって実行されているコマンド。 この情報は、データベースサーバーのコマンドラインに -zl オプションを指定して最後に作成された SQL 文の取得をオンにするか、sa_server_option システムプロシージャーを使用してこの機能をオンにした場合にのみ使用できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	ローを含むテーブルのオブジェクト ID。
record_id	BIGINT	関連付けられているローのロー ID。
owner	INT	待機しているロックを所有している接続の接続ハンドル。
is_victim	BIT	ロールバックされたトランザクションを識別。
rollback_operation_count	UNSIGNED INT	トランザクションをロールバックした場合に失われる可能性のある、コミットされていないオペレーションの数。

備考

log_deadlocks オプションが On に設定されている場合、データベースサーバーは、デッドロックに関する情報を内部バッファーに記録します。sa_report_deadlocks システムプロシージャーを使用して、ログ内の情報を表示できます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「システムイベント」『SQL Anywhere サーバー データベース管理』
- 「log_deadlocks オプション」『SQL Anywhere サーバー データベース管理』
- 「sa_server_option システムプロシージャー」 1170 ページ
- 「ブロックされているユーザーの判別」『SQL Anywhere サーバー SQL の使用法』
- 「-zl dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「sa_server_option システムプロシージャー」 1170 ページ

sa_reserved_words システムプロシージャー

予約語のリストを返します。SQL 文で使用するキーワードは、すべてではありませんが多くが予約語になっています。

構文

sa_reserved_words( )

備考

このプロシージャーはパラメーターを持たず、1 つのローにつき 1 つの語を返します。予約語のリストは、データベースファイルを作成するために使用されるソフトウェアのバージョンではなく、クエリを実行するデータベースサーバーのバージョンに基づきます。

パーミッション

なし。

関連する動作

なし。

参照

- 「予約語」1 ページ

例

次の文は、予約語のリストを返します。

```
SELECT *
FROM dbo.sa_reserved_words();
```

sa_reset_identity システムプロシージャー

次の identity の値をテーブルに設定できます。このシステムプロシージャーを使用して、次に挿入されるローの AUTOINCREMENT 値を変更します。

構文

sa_reset_identity(
tbl_name,
owner_name,
new_identity
)

引数

- ***tbl_name*** 識別値をリセットするテーブルを指定する CHAR(128) パラメーター。テーブルの所有者を指定しない場合、*tbl_name* はデータベースのテーブルをユニークに識別する必要があります。

- ***owner_name*** 識別値をリセットするテーブルの所有者を指定する CHAR(128) パラメーター。*owner_name* が指定されていないか NULL の場合、3 番目の引数に名前を付ける必要があります。次に例を示します。 CALL sa_reset_identity ('Employees', new_identity = 100);

- ***new_identity*** オートインクリメントの開始値を指定する BIGINT パラメーター。

備考

テーブルに挿入されるローに対して生成される次の ID 値は、*new_identity* + 1 です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

new_identity + 1 がテーブルの既存のローと競合するかどうかを確認するチェックは発生しません。たとえば、*new_identity* を 100 と指定した場合、挿入した次のローは 101 という ID 値を取得します。ただし、テーブルに 101 がすでに存在する場合、ローの挿入は失敗します。

sa_reset_identity システムプロシージャーは、AUTOINCREMENT または GLOBAL AUTOINCREMENT のデフォルトが指定されたカラムのないテーブルに対しては使用できません。

パーミッション

DBA 権限。

関連する動作

値が更新された後にチェックポイントを発生させます。

参照

- 「オートインクリメントデフォルト」『SQL Anywhere サーバー SQL の使用法』
- 「GLOBAL AUTOINCREMENT デフォルト」『SQL Anywhere サーバー SQL の使用法』

例

次の文は次の ID 値を 101 にリセットします。

```
CALL sa_reset_identity( 'Employees', 'GROUPO', 100 );
```

sa_rowgenerator システムプロシージャー

指定された開始値と終了値の間のローを格納した結果セットを返します。

構文

sa_rowgenerator(
[*rstart*
[, *rend*
[, *rstep*]]]
)

引数

- ***rstart*** 開始値を指定する任意の INTEGER パラメーター。デフォルト値は 0 です。
- ***rend*** *rstart* 以上の終了値を指定する任意の INTEGER パラメーター。デフォルト値は 100 です。
- ***rstep*** シーケンス値の増分を指定する任意の INTEGER パラメーター。デフォルト値は 1 です。

結果セット

カラム名	データ型	説明
row_num	INTEGER	シーケンス番号。

備考

sa_rowgenerator プロシージャーをクエリの FROM 句で使用して、番号のシーケンスを生成できます。RowGenerator システムテーブルを使用する代わりに、このプロシージャーを使用できます。sa_rowgenerator は次のようなタスクに使用できます。

- 結果セット内の既知の数のローについてテストデータを生成する。
- あらゆる範囲の値に対するローを格納した結果セットを生成する。たとえば、1 か月の毎日についてのローを生成したり、郵便番号の範囲を生成したりできます。
- 結果セット内に指定した数のローを格納するクエリを生成する。これは、クエリのパフォーマンスのテストに役立ちます。

正しい開始値、終了値、正の 0 以外の増分値を指定しないと、ローは返されません。

次の文を使用して、RowGenerator テーブルの動作をエミュレートできます。

```
SELECT row_num FROM sa_rowgenerator( 1, 255 );
```

パーミッション

なし。

関連する動作

なし。

参照

- 「RowGenerator テーブル (dbo)」1027 ページ
- 「集合関数では、NULL 値は無視されます。」『エラーメッセージ』

例

次のクエリは、現在の月の 1 日ごとに 1 つのローを含む結果セットを返します。

```
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;
```

次のクエリは、郵便番号が (0-9999)、(10000-19999)、...、(90000-99999) の範囲の地域に居住している従業員数を示します。従業員のいない範囲もあり、その場合は警告が発生します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

sa_rowgenerator プロシージャーを使用すると、ある範囲の郵便番号に従業員がいない場合でも、これらの範囲を生成できます。

```
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;
```

次の例は、データのローを 10 個生成し、それらを NewEmployees テーブルに挿入します。

```
INSERT INTO NewEmployees ( ID, Salary, Name )
SELECT row_num,
   CAST( RAND() * 1000 AS INTEGER ),
   'Mary'
FROM sa_rowgenerator( 1, 10 );
```

次の例は、sa_rowgenerator システムプロシージャーを使用して、すべての整数を含むビューを作成します。この例の値 2147483647 は、サポートされている最大の符号付き整数を表します。

```
CREATE VIEW Integers AS
SELECT row_num AS n
FROM sa_rowgenerator( 0, 2147483647, 1 );
```

次の例は、sa_rowgenerator システムプロシージャーを使用して、0001-01-01 ～ 9999-12-31 の日付を含むビューを作成します。この例の値 3652058 は、0001-01-01 ～ 9999-12-31 (それぞれサポートされている最初の日付と最後の日付) の日数を表します。

```
CREATE VIEW Dates AS
SELECT DATEADD( day, row_num, '0001-01-01' ) AS d
FROM sa_rowgenerator( 0, 3652058, 1 );
```

次のクエリは、1900 年から 2058 年までの間の 54 週あるすべての年を返します。

```
SELECT DATEADD ( day, row_num, '1900-01-01' ) AS d, DATEPART ( week, d ) w
FROM sa_rowgenerator ( 0, 63919, 1 )
WHERE w = 54;
```

sa_save_trace_data システムプロシージャー

トレーシングデータを基本テーブルに保存します。

構文

sa_save_trace_data( )

備考

トレーシングセッションが実行中に、診断データは診断トレーシングテーブルの一時的なバージョンに保存されます。トレーシングセッションを停止するときに、診断トレーシングの基本テーブルにトレーシングデータを永続的に保存するかどうかを指定します。データの保存を選択しない場合、sa_save_trace_data システムプロシージャーを使用して、セッションを停止した後にデータを保存することもできます。

sa_save_trace_data システムプロシージャーは、トレーシングが進行中の場合にエラーを返します。このシステムプロシージャーを使用するには、トレーシングを停止する必要があります。

トレーシングの停止時にユーザーが WITHOUT SAVING を指定した場合でも、sa_save_trace_data システムプロシージャーを使用できます。また、プロシージャーはトレーシングデータベースから呼び出す必要があります。

パーミッション

DBA 権限。

関連する動作

オートコミット。

参照

- 「診断トレーシングセッションの作成 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』
- 「診断トレーシングテーブル」1008 ページ

sa_send_udp システムプロシージャー

指定されたアドレスに UDP パケットを送信します。

構文

sa_send_udp(
destAddress,
destPort,
msg
)

引数

- ***destAddress*** ホスト名または IP アドレスを指定する CHAR(254) パラメーター。
- ***destPort*** 使用するポート番号を指定する UNSIGNED SMALLINT パラメーター。
- ***msg*** 指定されたアドレスに送信するメッセージを指定する LONG BINARY パラメーター。この値が文字列の場合は、一重引用符で囲みます。

備考

このプロシージャーは、指定されたアドレスに 1 つの UDP パケットを送信します。メッセージが正常に送信された場合は 0 を返し、エラーが発生した場合はエラーコードを返します。エラーコードは次のいずれかです。

- UDP ソケットで送信するにはメッセージが大きすぎる場合 (サイズはオペレーティングシステムによって決定)、または送信先アドレスに問題がある場合は -1
- オペレーティングシステムによって返される Winsock／Posix エラーコード

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

msg パラメーターにバイナリデータが含まれる場合、または文字列よりも複雑な場合は、変数を使用することもできます。次に例を示します。

```
CREATE VARIABLE v LONG BINARY;
SET v='This is a UDP message';
SELECT dbo.sa_send_udp( '10.25.99.124', 1234, v );
DROP VARIABLE v;
```

このプロシージャーを Mobile Link サーバーで開始される同期とともに使用して、Mobile Link リスナーユーティリティ (*dblsn.exe*) を起動できます。Mobile Link リスナーに通知する方法として sa_send_udp システムプロシージャーを使用する場合は、UDP パケットに 1 を追加してください。この数字は、サーバーで開始された同期プロトコル番号です。Mobile Link の将来のバージョンでは、新しいプロトコルバージョンによって Mobile Link リスナーの動作が変更される可能性があります。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_send_udp システムプロシージャーを使用した Push 通知の送信」『Mobile Link サーバー起動同期』

例

次の例は、メッセージ「This is a test」をポート 2345 の IP アドレス 10.25.99.196 に送信します。

```
CALL sa_send_udp( '10.25.99.196', 2345', 'This is a test' );
```

sa_server_messages システムプロシージャー

データベースサーバーメッセージウィンドウのメッセージを結果セットとして返すことができます。

構文

sa_server_messages([*first_msg*] [, *num_msgs*])

引数

- ***first_msg*** *num_msgs* パラメーターの符号に応じて、返される最初または最後のメッセージの ID を指定する任意の UNSIGNED BIGINT パラメーター。デフォルトは NULL です。つまり、*num_msgs* が NULL または正の場合、検索はリストの先頭から開始されます。*num_msgs* が負の場合、検索はリストの最後を過ぎたところから開始されます。

- ***num_msgs*** 返されるメッセージの数を指定する任意の BIGINT パラメーター。符号は、*first_msg* から始まるメッセージを要求するか、または *first_msg* で終わるメッセージを要求す

るかを示します。デフォルトは NULL です。つまり、*first_msg* からリストの最後までのすべてのメッセージが返されます。

結果セット

カラム名	データ型	説明
msg_id	UNSIGNED BIGINT	ユニークなメッセージ ID。メッセージ ID は 0 から始まります。
msg_text	LONG VARCHAR	メッセージテキスト。
msg_time	TIMESTAMP	メッセージが発行された時刻。
msg_severity	VARCHAR(255)	メッセージの重大度レベル。このカラムには、次のいずれかの値が含まれます。 ● **INFO** 情報メッセージ。 ● **WARN** 警告。 ● **ERR** エラー。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
msg_category	VARCHAR(255)	メッセージのカテゴリ。このカラムには、次のいずれかの値が含まれます。 ● **STARTUP/SHUTDOWN** データベースサーバーまたはデータベースの開始やシャットダウンに関連するメッセージ。 ● **CHKPT** チェックポイントに関連するメッセージ。 ● **MSG** MESSAGE 文または PRINT 文を使用して生成されたメッセージ。 ● **DBA_MSG** DBA パーミッションを必要とする MESSAGE 文を使用して生成されたメッセージ。イベントログに送信されたメッセージなど。 ● **CONN** データベースサーバーのコネクティビティに関するメッセージ。 ● **OTHER** その他のタイプのすべてのメッセージ。
msg_database	VARCHAR(255)	メッセージが特定の 1 つのデータベースに適用される場合は、そのメッセージに関連付けられたデータベース名。それ以外の場合は NULL。

備考

新しいメッセージがコンソールに送信される場合、メッセージの数が MessageCategoryLimit プロパティの値を超えていると、同じカテゴリまたは同じ重大度レベルの古いメッセージは削除されます。その結果、結果セットにギャップができ、連続した 2 つのローでメッセージ ID が不連続となる場合があります。

STARTUP/SHUTDOWN メッセージカテゴリでは、サーバーのシャットダウンメッセージは表示されません。シャットダウンメッセージは、サーバーで複数のデータベースが実行中で、そのうちの 1 つ以上がシャットダウンされた場合のみ表示されます。

パーミッション

なし。

関連する動作

なし。

参照

- MessageCategoryLimit サーバープロパティ『SQL Anywhere サーバー データベース管理』

例

次のコマンドは、ID が 3 のメッセージを先頭として、100 メッセージを要求します。

```
CALL sa_server_messages( 3, 100 );
```

次のコマンドは、メッセージ 4032 までの (このメッセージを含む) 500 メッセージを要求します。

```
CALL sa_server_messages( 4032, -500 );
```

次のコマンドは、メッセージ 3 以降のすべてのメッセージを要求します。

```
CALL sa_server_messages( 3, NULL );
CALL sa_server_messages( 3 );
```

次のコマンドは、リストの先頭からの 100 メッセージを要求します。

```
CALL sa_server_messages( NULL, 100 );
```

次のコマンドは、リストの最後の 100 メッセージを要求します。

```
CALL sa_server_messages( NULL, -100 );
```

次のコマンドは、リストのすべてのメッセージを要求します。

```
CALL sa_server_messages( NULL, NULL );
CALL sa_server_messages( );
```

sa_server_option システムプロシージャー

サーバーの実行中に、サーバーオプションを上書きします。

構文

sa_server_option(
opt,
val
)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

引数

- ***opt*** サーバーオプション名を指定する CHAR(128) パラメーター。
- ***val*** サーバーオプションの新しい値を指定する CHAR(128) パラメーター。

備考

データベース管理者はこのプロシージャーを使用して、データベースサーバーを再起動しないでデータベースサーバーオプションの一部を一時的に上書きできます。

このプロシージャーを使用して変更されるオプション値は、サーバーが停止するとデフォルト値にリセットされます。サーバーが起動するたびにオプション値を変更する場合は、データベースサーバーの起動時に対応するデータベースサーバーオプションがあればそれを指定できます (これらのオプションは次の表で一番右のカラムにリストされています)。

次のオプション設定を変更できます。

オプション名	値	デフォルト	サーバーオプション	参照
AutoMultiProgrammingLevel	YES、NO	YES	「-gna dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	「マルチプログラミングレベルのデータベースサーバー設定」『SQL Anywhere サーバー データベース管理』
AutoMultiProgrammingLevelStatistics	YES、NO	NO	「-gns dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	AutoMultiProgrammingLevelStatistics サーバープロパティ『SQL Anywhere サーバー データベース管理』
CacheSizingStatistics	YES、NO	NO	「-cs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	CacheSizingStatistics サーバープロパティ『SQL Anywhere サーバー データベース管理』
CollectStatistics	YES、NO	YES	「-k dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	CollectStatistics サーバープロパティ『SQL Anywhere サーバー データベース管理』

オプション名	値	デフォルト	サーバーオプション	参照
ConnsDisabled	YES、NO	NO		ConnsDisabled サーバープロパティ『SQL Anywhere サーバー データベース管理』
ConnsDisabledForDB	YES、NO	NO		
ConsoleLogFile	*filename*		「-o dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	ConsoleLogFile サーバープロパティ『SQL Anywhere サーバー データベース管理』
ConsoleLogMaxSize	バイト単位での*ファイルサイズ*		「-on dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	ConsoleLogMaxSize サーバープロパティ『SQL Anywhere サーバー データベース管理』
CurrentMultiProgrammingLevel	整数	20	「-gn dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	「マルチプログラミングレベルのデータベースサーバー設定」『SQL Anywhere サーバー データベース管理』
DatabaseCleaner	ON、OFF	ON		DatabaseCleaner データベースプロパティ『SQL Anywhere サーバー データベース管理』
DeadlockLogging	ON、OFF、RESET、CLEAR	OFF	「log_deadlocks オプション」『SQL Anywhere サーバー データベース管理』	

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプション名	値	デフォルト	サーバーオプション	参照
DebuggingInformation	YES、NO	NO	「-z dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	DebuggingInformation サーバープロパティ『SQL Anywhere サーバー データベース管理』
DropBadStatistics	YES、NO	YES		
DropUnusedStatistics	YES、NO	YES		
IdleTimeout	整数 (分)	240	「-ti dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	IdleTimeout サーバープロパティ『SQL Anywhere サーバー データベース管理』
IPAddressMonitorPeriod	整数 (秒)	ポータブルデバイスの場合は 120、それ以外の場合は 0	「-xm dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	IPAddressMonitorPeriod サーバープロパティ『SQL Anywhere サーバー データベース管理』
LivenessTimeout	整数 (秒)	120	「-tl dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	LivenessTimeout サーバープロパティ『SQL Anywhere サーバー データベース管理』
MaxMultiProgrammingLevel	整数	CurrentMultiProgrammingLevel 値の 4 倍	「-gnh dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	「マルチプログラミングレベルのデータベースサーバー設定」『SQL Anywhere サーバー データベース管理』

オプション名	値	デフォルト	サーバーオプション	参照
MessageCategoryLimit	整数	400		MessageCategoryLimit サーバープロパティ『SQL Anywhere サーバー データベース管理』
MinMultiProgrammingLevel	整数	-gtc サーバーオプションの値の最小値とコンピューター上の論理 CPU の数	「-gnl dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	「マルチプログラミングレベルのデータベースサーバー設定」『SQL Anywhere サーバー データベース管理』
OptionWatchAction	MESSAGE、ERROR	MESSAGE		● 「オプション設定のモニタリング」『SQL Anywhere サーバー データベース管理』 ● OptionWatchAction データベースプロパティ『SQL Anywhere サーバー データベース管理』
OptionWatchList	カンマで区切られたデータベースオプションのリスト			● 「オプション設定のモニタリング」『SQL Anywhere サーバー データベース管理』 ● OptionWatchList データベースプロパティ『SQL Anywhere サーバー データベース管理』
ProcedureProfiling	YES、NO、RESET、CLEAR	NO		ProcedureProfiling データベースプロパティ『SQL Anywhere サーバー データベース管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプション名	値	デフォルト	サーバーオプション	参照
ProfileFilterConn	*connection-id*			ProfileFilterConn サーバープロパティ『SQL Anywhere サーバー データベース管理』
ProfileFilterUser	*user-id*			ProfileFilterUser サーバープロパティ『SQL Anywhere サーバー データベース管理』
QuittingTime	有効な日付と時刻		「-tq dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	QuittingTime サーバープロパティ『SQL Anywhere サーバー データベース管理』
RememberLastPlan	YES、NO	NO	「-zp dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	RememberLastPlan サーバープロパティ『SQL Anywhere サーバー データベース管理』
RememberLastStatement	YES、NO	NO	「-zl dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	RememberLastStatement サーバープロパティ『SQL Anywhere サーバー データベース管理』
RequestFilterConn	*connection-id*, -1			RequestFilterConn サーバープロパティ『SQL Anywhere サーバー データベース管理』

オプション名	値	デフォルト	サーバーオプション	参照
RequestFilterDB	*database-id*, -1			RequestFilterDB サーバープロパティ『SQL Anywhere サーバー データベース管理』
RequestLogFile	*filename*		「-zo dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	RequestLogFile サーバープロパティ『SQL Anywhere サーバー データベース管理』
RequestLogging	SQL、HOSTVARS、PLAN、PROCEDURES、TRIGGERS、OTHER、BLOCKS、REPLACE、ALL、YES、NONE、NO	NONE	「-zr dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	RequestLogging サーバープロパティ『SQL Anywhere サーバー データベース管理』
RequestLogMaxSize	バイト単位での*ファイルサイズ*		「-zs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	RequestLogMaxSize サーバープロパティ『SQL Anywhere サーバー データベース管理』
RequestLogNumFiles	整数		「-zn dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	RequestLogNumFiles サーバープロパティ『SQL Anywhere サーバー データベース管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプション名	値	デフォルト	サーバーオプション	参照
RequestTiming	YES、NO	NO	「-zt dbeng12/dbsrv12 オプション」『SQL Anywhere サーバー データベース管理』	RequestTiming サーバープロパティ『SQL Anywhere サーバー データベース管理』
SecureFeatures	*feature-list*		「-sf dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	「機能の保護」『SQL Anywhere サーバー データベース管理』
StatisticsCleaner	ON、OFF	ON		
WebClientLogFile	*filename*		「-zoc dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	WebClientLogFile サーバープロパティ『SQL Anywhere サーバー データベース管理』
WebClientLogging	ON、OFF	OFF	「-zoc dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』	WebClientLogging サーバープロパティ『SQL Anywhere サーバー データベース管理』

- **AutoMultiProgrammingLevel**　YES に設定すると、データベースサーバーでマルチプログラミングレベルが自動的に調整されます。マルチプログラミングレベルとは、一度にアクティブにできるタスクの最大数を制御するものです。このオプションを NO に設定して、マルチプログラミングレベルを手動で制御するようにした場合、マルチプログラミングレベルの初期値、最小値、最大値を設定できます。

- **AutoMultiProgrammingLevelStatistics**　YES に設定すると、マルチプログラミングレベルの自動調整の統計がデータベースサーバーのメッセージログに表示されます。

- **CacheSizingStatistics**　YES に設定した場合、キャッシュサイズが変更されるたびに、データベースサーバーメッセージウィンドウにキャッシュ情報を表示します。

- **CollectStatistics**　YES に設定すると、データベースサーバーはパフォーマンスモニターの統計情報を収集します。
- **ConnsDisabled**　YES に設定すると、データベースサーバー上のデータベースに対する他の接続は許可されません。
- **ConnsDisabledForDB**　YES に設定すると、その他の接続が現在のデータベースに許可されます。
- **ConsoleLogFile**　データベースサーバーメッセージログ情報の記録に使用されるファイル名。空の文字列を指定すると、ファイルへのロギングが停止します。これは SQL 文字列なので、パスの円記号は 2 つ重ねます。
- **ConsoleLogMaxSize**　データベースサーバーメッセージログ情報の記録に使用されるファイルの最大サイズ (バイト単位)。データベースサーバーメッセージログファイルが、このプロパティまたは -on サーバーオプションで指定されたサイズに達すると、ファイルが拡張子 *.old* の付いた名前に変更されます (既存のファイルが存在する場合は、同じ名前で置換されます)。その後、データベースサーバーメッセージログファイルが再開されます。
- **CurrentMultiProgrammingLevel**　データベースサーバーのマルチプログラミングレベルを設定します。
- **DatabaseCleaner**　このオプションの設定は、iAnywhere テクニカルサポートの指示があった場合を除いて、変更しないでください。
- **DeadlockLogging**　デッドロックのロギングを制御します。値 deadlock_logging もサポートされます。デッドロックロギングオプションは、Sybase Central の **[データベースのプロパティ]** ウィンドウでも設定できます。次の値がサポートされています。
  - **ON**　デッドロックロギングを有効にします。
  - **OFF**　デッドロックロギングを無効にしますが、デッドロックデータの表示は可能です。
  - **RESET**　デッドロックデータが存在する場合は、記録済みのデッドロックデータをクリアし、その後、デッドロックロギングを有効にします。
  - **CLEAR**　デッドロックデータが存在する場合は、記録済みのデッドロックデータをクリアし、その後、デッドロックロギングを無効にします。

  デッドロックロギングが有効になると、sa_report_deadlocks システムプロシージャーを使用してデータベースからデッドロック情報を取得することができます。
- **DebuggingInformation**　診断メッセージなどのメッセージをトラブルシューティングのために表示します。メッセージは、データベースサーバーメッセージウィンドウに表示されます。
- **DropBadStatistics**　自動統計管理で、不適切な推定値を返す統計をデータベースから削除できるようにします。
- **DropUnusedStatistics**　自動統計管理で、連続 90 日間使用されなかった統計をデータベースから削除できるようにします。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **IdleTimeout**　minutes で指定された時間の間、要求を送信しなかった TCP/IP 接続を切断します。こうすることで、アクティブではない接続がロックを無制限に維持することが回避されます。

- **IPAddressMonitorPeriod**　新しい IP アドレスをチェックする時間を秒単位で設定します。最小値は 10、デフォルト値は 0 です。ポータブルデバイスの場合、デフォルト値は 120 秒です。

- **LivenessTimeout**　接続が維持されていることを確認するため、クライアント／サーバーの TCP/IP ネットワークを介して、定期的に活性パケットが送信されます。ネットワークサーバーが、活性パケットを検出することなく、LivenessTimeout 時間にわたって実行されると、通信は切断されます。

- **MaxMultiProgrammingLevel**　データベースサーバーのマルチプログラミングレベルの最大値を設定します。

- **MessageCategoryLimit**　sa_server_messages システムプロシージャーを使用して取り出すことのできるメッセージの最小数を、重大度レベルとカテゴリごとに設定します。

- **MinMultiProgrammingLevel**　データベースサーバーのマルチプログラミングレベルの最小値を設定します。

- **OptionWatchAction**　リストにオプションを設定しようとしたときにデータベースサーバーが実行するアクションを指定します。サポートされる値は MESSAGE と ERROR です。OptionWatchAction が MESSAGE に設定されており、OptionWatchList によって指定されるオプションが設定されている場合は、データベースサーバーメッセージウィンドウにメッセージが表示され、設定されているオプションがオプションウォッチリストに入っていることが示されます。

  OptionWatchAction が ERROR に設定されている場合は、オプションウォッチリストにオプションが入っているためにオプションを設定できないことを示すエラーが返されます。

  このプロパティの現在の設定は、次のクエリを実行して表示できます。

  ```
  SELECT DB_PROPERTY( 'OptionWatchAction' );
  ```

- **OptionWatchList**　設定したときに通知の対象としたり、データベースサーバーでエラーを返したりするデータベースオプションのリストを、カンマで区切って指定します。この文字列の長さは 128 バイトに制限されています。デフォルトでは、空の文字列です。たとえば、次のコマンドは、ウォッチするオプションのリストに automatic_timestamp、float_as_double、tsql_hex_constant の各オプションを追加します。

  ```
  CALL sa_server_option( 'OptionWatchList','automatic_timestamp,
                          float_as_double,tsql_hex_constant' )
  ```

  このプロパティの現在の設定は、次のクエリを実行して表示できます。

  ```
  SELECT DB_PROPERTY( 'OptionWatchList' );
  ```

- **ProcedureProfiling**　ストアドプロシージャー、ファンクション、イベント、トリガーについてのプロシージャープロファイリングを制御します。プロシージャープロファイリングは、ストアドプロシージャー、ファンクション、イベント、トリガーの実行所要時間を示し

ます。Sybase Central の **[データベースのプロパティ]** ウィンドウでも、プロシージャープロファイリングオプションを設定できます。

○ **YES**　現在の接続先データベースについてプロシージャープロファイリングを有効にします。

○ **NO**　プロシージャープロファイリングを無効にしますが、プロファイリングデータの表示は可能です。

○ **RESET**　YES または NO の設定は変更しないで、プロファイリングカウンターを 0 に戻します。

○ **CLEAR**　プロファイリングカウンターを 0 に戻し、プロシージャープロファイリングを無効にします。

プロファイリングを有効にすると、システムプロシージャー sa_procedure_profile_summary と sa_procedure_profile を使用して、データベースからプロファイリング情報を取り出すことができます。

- **ProfileFilterConn**　他の接続がデータベースを使用する処理を阻害することなく、特定の接続 ID に関するプロファイル情報を取得するように、データベースサーバーに指示します。接続フィルターが有効な場合、SELECT PROPERTY('ProfileFilterConn') の戻り値は監視されている接続の接続 ID です。ID が指定されていない場合、または接続フィルターが無効な場合、戻り値は -1 です。

- **ProfileFilterUser**　データベースサーバーに、特定のユーザー ID のプロファイリング情報を取得するよう指示します。

- **QuittingTime**　データベースサーバーに、指定された時間にサーバーを停止するよう指示します。

- **RememberLastPlan**　接続で最後に実行されたクエリの長いテキストプランをキャプチャするように、データベースサーバーに指示します。この設定は、-zp サーバーオプションでも制御されます。

  RememberLastPlan が ON の場合は、LastPlanText 接続プロパティの値を問い合わせることで、この接続で最後に実行されたクエリのプランのテキスト表現を取得できます。

  ```
  SELECT CONNECTION_PROPERTY( 'LastPlanText' );
  ```

- **RememberLastStatement**　データベースサーバーに、サーバー上で実行されている各データベースに関して最後に作成された SQL 文を取得するように指示します。ストアドプロシージャーコールの場合、プロシージャー内の文ではなく、最も外側のプロシージャーコールのみが表示されます。

  RememberLastStatement が ON の場合は、LastStatement 接続プロパティの値を問い合わせることで、接続に関する LastStatement の現在の値を取得できます。

  ```
  SELECT CONNECTION_PROPERTY( 'LastStatement' );
  ```

  クライアントでの文のキャッシュが有効であり、キャッシュされた文が再使用されているとき、このプロパティは空の文字列を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

RememberLastStatement が ON の場合、次の文は指定された接続に対して最後に作成された文を返します。

```
SELECT CONNECTION_PROPERTY( 'LastStatement', connection-id );
```

sa_conn_activity システムプロシージャーは、すべての接続に対して同じ情報を返します。

> **警告**
> -zl が指定されている場合、または RememberLastStatement サーバー設定がオンになっている場合、すべてのユーザーが sa_conn_activity システムプロシージャーを呼び出すか、LastStatement 接続プロパティの値を取得することにより、他のユーザーが最後に作成した SQL 文を見つけることができます。このオプションは注意して使用し、不要な場合はオフにしてください。

- **RequestFilterConn**　要求ログ情報をフィルターして、特定の接続の情報のみ記録されるようにします。これによって、多くのアクティブな接続または複数のデータベースのあるデータサーバーを監視するときに、要求ログファイルのサイズを削減できます。次の文を実行して、接続 ID を取得できます。

  ```
  CALL sa_conn_info( );
  ```

  接続 ID を取得した後でロギングする特定の接続を指定するには、次の文を実行します。

  ```
  CALL sa_server_option( 'RequestFilterConn', connection-id );
  ```

  フィルターは、明示的にリセットされるか、データベースサーバーが停止するまで有効なままになります。フィルターをリセットするには、次の文を使用します。

  ```
  CALL sa_server_option( 'RequestFilterConn', -1 );
  ```

- **RequestFilterDB**　要求ログ情報をフィルターして、特定のデータベースの情報のみ記録されるようにします。これによって、複数のデータベースのあるサーバーを監視するときに、要求ログファイルのサイズを削減できます。目的のデータベースに接続しているときに次の文を実行して、データベース ID を取得できます。

  ```
  SELECT CONNECTION_PROPERTY( 'DBNumber' );
  ```

  特定のデータベースについての情報のみ記録することを指定するには、次の文を実行します。

  ```
  CALL sa_server_option( 'RequestFilterDB', database-id );
  ```

  フィルターは、明示的にリセットされるか、データベースサーバーが停止するまで有効なままになります。フィルターをリセットするには、次の文を使用します。

  ```
  CALL sa_server_option( 'RequestFilterDB', -1 );
  ```

- **RequestLogFile**　要求情報の記録に使用されるファイルの名前。空の文字列を指定すると、要求ログファイルへのロギングが停止します。要求のロギングが有効でも、要求のログファイルを指定しなかった場合、または空の文字列に設定されている場合、サーバーは要求をデータベースサーバーメッセージウィンドウにロギングします。SQL 文字列と同じように、パスの円記号は 2 つ重ねます。

クライアントでの文のキャッシュが有効であり、かつ、ログが *tracetime.pl* Perl スクリプトを使用して分析される場合は、要求ログが取得される間、max_client_statements_cached オプションを 0 に設定して、クライアントでの文のキャッシュを無効にする必要があります。

- **RequestLogging**　データベースサーバーオプション -zr と -zo とともにトラブルシューティングで使用するためにデータベースサーバーに送信される個々の SQL 文のロギングを有効にします。次の値をプラス記号 (+) またはカンマで区切って組み合わせた値です。

  - **PLAN**　実行プランのロギングを有効にします (短期)。プロシージャーの実行プランは、プロシージャー (PROCEDURES) のロギングが有効な場合にも記録されます。

  - **HOSTVARS**　ホスト変数の値のロギングを有効にします。HOSTVARS を指定した場合、SQL にリストされている情報もロギングされます。

  - **PROCEDURES**　プロシージャー内から実行されている文のロギングを有効にします。

  - **TRIGGERS**　トリガー内から実行されている文のロギングを有効にします。

  - **OTHER**　SQL に含まれないその他の要求タイプのロギングを有効にします (FETCH や PREFETCH など)。ただし、OTHER を指定して SQL を指定しない場合、SQL+OTHER を指定した場合と同じです。OTHER を含めると、ログファイルが急速に拡大し、サーバーのパフォーマンス低下につながることがあります。

  - **BLOCKS**　別の接続で接続がブロックされたときと、接続のブロックが解除されたときに表示する詳細のロギングを有効にします。

  - **REPLACE**　ロギングの開始時に、既存の要求ログは同じ名前を持つ新規の (空の) ログで置換されます。それ以外の場合、既存の要求ログが開き、新規エントリがファイルの末尾に追加されます。

  - **ALL**　すべてのサポート情報をロギングします。これは、SQL+PLAN+HOSTVARS +PROCEDURES+TRIGGERS+OTHER+BLOCKS を指定した場合と同じです。この設定では、ログファイルが急速に拡大し、サーバーのパフォーマンス低下につながることがあります。

  - **NO または NONE**　要求ログに対するロギングを無効にします。

  このプロパティの現在の設定は、次のクエリを実行して表示できます。

  ```
  SELECT PROPERTY( 'RequestLogging' );
  ```

  詳細については、「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』を参照してください。

- **RequestLogMaxSize**　要求ログ情報の記録に使用されるファイルのバイト単位での最大サイズ。0 を指定した場合は、要求ロギングファイルの最大サイズは適用されず、ファイルの名前は変更されません。これはデフォルト値です。

  要求ログファイルが、sa_server_option システムプロシージャーまたは -zs サーバーオプションで指定されたサイズに達すると、ファイルが拡張子 *.old* の付いた名前に変更されます (既

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

存のファイルが存在する場合は、同じ名前で置換されます)。要求ログファイルが再起動します。

- **RequestLogNumFiles**　保持する要求ログファイルのコピーの数。

  要求ロギングが長時間にわたって有効になっていると、要求ログファイルが大きくなることがあります。-zn オプションを使用すると、保持する要求ログファイルのコピー数を指定できます。

- **RequestTiming**　データベースサーバーに、各接続のタイミング情報を管理するように指示します。この機能はデフォルトでオフになっています。オンにすると、データベースサーバーは各接続の累積タイマを管理します。このタイマは、接続が確立した状態でのサーバーステータスごとの時間を示すものです。sa_performance_diagnostics システムプロシージャーを使用して、このタイミング情報の概要を取得できます。または、次の接続プロパティを調べて各値を取得できます。

  - ReqCountUnscheduled
  - ReqTimeUnscheduled
  - ReqCountActive
  - ReqTimeActive
  - ReqCountBlockIO
  - ReqTimeBlockIO
  - ReqCountBlockLock
  - ReqTimeBlockLock
  - ReqCountBlockContention
  - ReqTimeBlockContention

  RequestTiming サーバープロパティがオンであると、追加のカウンターを管理するため、要求ごとに多少のオーバーヘッドがかかります。

- **SecureFeatures**　すでに動作しているデータベースサーバーの保護機能を有効または無効にできます。*feature-list* は、機能名または機能セットのカンマ区切りリストです。このリストに機能を追加することで、追加した機能の可用性を制限します。保護機能のリストから項目を削除するには、保護機能名の前にマイナス記号 (-) を指定します。

  機能を有効または無効にするために行った変更は、接続で直ちに有効になります。この設定は、sa_server_option システムプロシージャーを実行する接続に影響を与えません。変更を確認するには、切断してから再接続する必要があります。

  注意
  sa_server_option システムプロシージャーを使用して、機能を有効または無効にするには、データベースサーバーの起動時に -sk オプションを使用してキーを指定してから、secure_feature_key データベースオプションの値を -sk で指定したキーに設定します (たとえば、SET TEMPORARY OPTION secure_feature_key = 'j978kls12' など)。secure_feature_key データベースオプションを -sk 値に設定すると、保護機能の設定を変更できます。

  たとえば、2 つの機能を無効にして 3 番目を有効にするには、次の構文を使用します。

  ```
  CALL sa_server_option('SecureFeatures', 'CONSOLE_LOG,WEBCLIENT_LOG,-
  REQUEST_LOG' );
  ```

この文を実行すると、CONSOLE_LOG と WEBCLIENT_LOG が保護機能のリストに追加され、REQUEST_LOG がリストから削除されます。

- **StatisticsCleaner**　統計クリーナーでは、テーブルに対してスキャンを実行して、不適切な推定値を示す統計を修正します。デフォルトでは、統計クリーナーはバックグラウンドで実行され、パフォーマンスへの影響は最小限に抑えられます。

  統計クリーナーをオフにしても統計ガバナーは無効になりませんが、統計クリーナーがオフのときには、統計はクエリの実行時にのみ作成または修正されます。

- **WebClientLogFile**　Web サービスクライアントログファイルの名前。Web サービスクライアントログファイルは、-zoc サーバーオプションや WebClientLogFile プロパティを使用して、ファイル名を設定またはリセットするたびにトランケートされます。これは文字列であるため、パスの円記号は 2 つ重ねます。

- **WebClientLogging**　このオプションは、Web サービスクライアントのロギングを有効または無効にします。ログに記録される情報には、HTTP の要求と応答のデータが含まれています。ON を指定すると Web サービスクライアントログファイルへのロギングが開始され、OFF を指定するとファイルへのロギングが中止されます。

パーミッション

アプリケーションプロファイリングまたは要求ロギングに関連する次のオプションには、DBA 権限または PROFILE 権限が必要です。

- ProcedureProfiling
- ProfileFilterConn
- ProfileFilterUser
- RequestFilterConn
- RequestFilterDB
- RequestLogFile
- RequestLogging
- RequestLogMaxSize
- RequestLogNumFiles

この他のすべてのオプションには DBA 権限が必要です。

関連する動作

なし。

参照

- 「システムプロシージャーを使用したプロシージャープロファイリング」『SQL Anywhere サーバー SQL の使用法』
- 「カラム統計の更新」『SQL Anywhere サーバー SQL の使用法』
- 「sa_performance_diagnostics システムプロシージャー」1140 ページ
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「log_deadlocks オプション」『SQL Anywhere サーバー データベース管理』
- 「secure_feature_key」『SQL Anywhere サーバー データベース管理』
- 「-sk dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
- 「-sf dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の文は、データベースサーバーへの新しい接続を禁止します。

```
CALL sa_server_option( 'ConnsDisabled', 'YES' );
```

次の文は、現在のデータベースへの新しい接続を禁止します。

```
CALL sa_server_option( 'ConnsDisabledForDB', 'YES' );
```

次の文は、すべての SQL 文、プロシージャーの呼び出し、プラン、イベントのブロックとブロック解除のロギングを有効にし、新規要求ログの開始を指定します。

```
CALL sa_server_option( 'RequestLogging', 'SQL+PROCEDURES+BLOCKS+PLAN+REPLACE' );
```

sa_set_http_header システムプロシージャー

Web サービスによる HTTP 応答ヘッダーの設定を許可します。

構文

sa_set_http_header(
fldname,
val
)

引数

- ***fldname*** HTTP ヘッダーフィールドのいずれかの名前を含む文字列を指定する CHAR(128) パラメーター。
- ***val*** 指定されたパラメーターに設定する値を指定する LONG VARCHAR パラメーター。応答ヘッダーを NULL に設定すると、そのヘッダーは事実上削除されます。

備考

特別なヘッダーフィールド @HttpStatus を設定すると、要求によってステータスコードが返されるように設定されます。ステータスコードは応答コードとも呼ばれます。たとえば、次のスクリプトはステータスコードを 404 Not Found に設定します。

```
CALL sa_set_http_header( '@HttpStatus', '404' );
```

3 桁のステータスコードを指定し、コロンで区切ったテキストメッセージを任意で追加することで、ユーザー定義のステータスメッセージを作成できます。たとえば、次のスクリプトは、メッセージ付きのステータスコード "999 User Code" を出力します。

```
CALL sa_set_http_header( '@HttpStatus', '999:User Code' );
```

注意
LogOptions プロトコルオプションを使用してログに記録している場合、ユーザー定義のステータステキストメッセージはデータベースの文字セットには変換されません。

エラーメッセージの本文は、自動的に挿入されます。有効な HTTP エラーコードだけが使用できます。ステータスに無効なコードを設定すると、SQL エラーが発生します。

sa_set_http_header プロシージャーは、呼び出されたときに、ヘッダーフィールドの既存のヘッダー値を必ず上書きします。

データベースサーバーによって自動的に生成された応答ヘッダーは削除できます。たとえば、次のコマンドは Expires 応答ヘッダーを削除します。

```
CALL sa_set_http_header( 'Expires', NULL );
```

パーミッション

なし。

関連する動作

なし。

参照

- 「NEXT_HTTP_RESPONSE_HEADER 関数 [Web サービス]」291 ページ
- 「HTTP_RESPONSE_HEADER 関数 [Web サービス]」251 ページ
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ
- 「LogOptions (LOPT) プロトコルオプション」『SQL Anywhere サーバー データベース管理』

例

次の例は、Content-Type ヘッダーフィールドを text/html に設定します。

```
CALL sa_set_http_header( 'Content-Type', 'text/html' );
```

sa_set_http_option システムプロシージャー

Web サービスによるプロセス制御用の HTTP オプションの設定を許可します。

構文

sa_set_http_option(
optname,
val
)

引数

- ***optname*** HTTP オプションのいずれかの名前を含む文字列を指定する CHAR(128) パラメーター。

  サポートされるオプションは次のとおりです。

  - **CharsetConversion** このオプションを使用して、結果セットをデータベースの文字セットエンコードからクライアントの文字セットエンコードに自動的に変換するかどうかを制御します。指定できる値は、ON と OFF だけです。デフォルト値は ON です。

○ **AcceptCharset**　このオプションを使用して、応答の文字セットエンコードに関する Web サーバーの優先度を指定します。1 つ以上の文字セットエンコードを優先度順に指定できます。このオプションの構文は、RFC2616 Hypertext Transfer Protocol の HTTP Accept-Charset 要求ヘッダーフィールドの仕様に使用する構文に準拠します。

Web ブラウザーなどの HTTP クライアントでは、Accept-Charset 要求ヘッダーを使用して、優先度順の文字セットエンコードのリストを指定できます。必要に応じて、各エンコードには、対応する品質値 (q=*qvalue*) を指定できます。この品質値は、そのエンコードに対するクライアントの優先度を表します。デフォルトでは、品質値は 1 (q=1) です。次に例を示します。

```
Accept-Charset: iso-8859-5, utf-8;q=0.8
```

AcceptCharset HTTP オプション値内のプラス記号 (+) は、現在のデータベースの文字セットエンコードを表すショートカットとして使用できます。また、プラス記号は、クライアントのリストでもエンコードが指定されている場合に、クライアントによって割り当てられている品質値に関係なく、データベースの文字セットエンコードを優先する必要があることも示します。

AcceptCharset HTTP オプション内のアスタリスク (*) は、クライアントとサーバーのリストに共通部分がない場合に、クライアントで優先される文字セットエンコードがサーバーでもサポートされていれば、Web サービスでその文字セットエンコードを使用する必要があることを示します。

応答の送信時には、クライアントと Web サービスの両方で優先される最初の文字セットエンコードが使用されます。クライアントの優先度順が最初に考慮されます。共通のエンコード優先度が存在しない場合は、Web サービスのリストにアスタリスク (*) がなければ、Web サービスで最も優先されるエンコードが使用されます。アスタリスクがある場合は、クライアントで最も優先されるエンコードが使用されます。

AcceptCharset HTTP オプションを使用しない場合は、クライアントで指定され、サーバーでサポートされている最も優先度の高い文字セットエンコードが使用されます。クライアントで指定されたどのエンコードもサポートされていない場合 (またはクライアントが Accept-Charset 要求ヘッダーを送信しない場合) は、データベースの文字セットエンコードが使用されます。

クライアントが Accept-Charset ヘッダーを送信しない場合は、次のいずれかのアクションが実行されます。

- AcceptCharset HTTP オプションが指定されていない場合、Web サーバーではデータベースの文字セットエンコードを使用します。
- AcceptCharset HTTP オプションが指定されている場合、Web サーバーでは最も優先度の高い文字セットエンコードを使用します。

クライアントが Accept-Charset ヘッダーを送信する場合は、次のいずれかのアクションが実行されます。

- AcceptCharset HTTP オプションが指定されていない場合、Web サーバーではクライアントで優先される文字セットエンコードのいずれかを使用しようとします (最も優先度の

高いエンコードから開始)。クライアントで優先されるどのエンコードも Web サーバーでサポートされていない場合、Web サーバーではデータベースの文字セットエンコードを使用します。

- AcceptCharset HTTP オプションが指定されている場合、Web サーバーでは両方の優先度リストで共通する最初の文字セットエンコードを使用しようとします (クライアントで最も優先されるエンコードから開始)。たとえば、クライアントが送信する Accept-Charset ヘッダーリストのエンコードの優先度順が iso-a、iso-b、iso-c であり、Web サーバーでの優先度順が iso-b、iso-a、iso-c の場合、iso-a が選択されます。

  Web client: iso-a, iso-b, iso-c
  Web server: iso-b, iso-a, iso-c

  2 つのリストに共通部分がない場合、Web サーバーで最初に優先される文字セットが使用されます。次の例では、エンコード iso-d が使用されます。

  Web client: iso-a, iso-b, iso-c
  Web server: iso-d, iso-e, iso-f

  AcceptCharset HTTP オプションにアスタリスク ('*') が含まれている場合は、クライアントのエンコードが優先されるため、iso-a が使用されます。基本的に、アスタリスクを使用すると、2 つのリストに共通部分がないという状況がなくなります。

理想的な状況は、クライアントと Web サービスの両方でデータベースの文字セットエンコードが使用される場合です。このような場合、文字セットの変換の必要性がなくなり、Web サーバーの応答時間が向上します。

CharsetConversion オプションを OFF に設定している場合、AcceptCharset の処理は実行されないことに注意してください。

○ **SessionID**　このオプションを使用して、HTTP セッションの作成、削除、または名前の変更を行います。Web サービスでこのオプションを設定して HTTP セッションを作成するときにはデータベース接続は維持されますが、サーバーの再起動時にはセッションは維持されません。すでにセッションコンテキスト内にある場合、この呼び出しによってセッションの名前が新しいセッション ID に変更されます。NULL 値を使用して呼び出した場合、Web サービスが終了するとセッションは削除されます。

生成されるセッションキーの長さは 128 文字に制限され、複数のデータベースがロードされている場合はデータベース間でユニークになります。

○ **SessionTimeout**　このオプションを使用して、非アクティブ状態の HTTP セッションを維持する時間を分単位で指定します。このタイムアウト期間は、HTTP 要求で指定のセッションが使用されると常にリセットされます。SessionTimeout を過ぎると、セッションは自動的に削除されます。

● ***val***　指定されたオプションに設定する値を指定する LONG VARCHAR パラメーター。

備考

Web サービスを処理する文またはプロシージャー内でこのプロシージャーを使用して、オプションを設定します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

sa_set_http_option が Web サービスによって開始されたプロシージャー内から呼び出され、オプションまたはオプションの値のいずれかが無効な場合は、エラーが返ります。

パーミッション

なし。

関連する動作

なし。

参照

- 「文字セットの変換に関する考慮事項」『SQL Anywhere サーバー プログラミング』
- 「HTTP サーバーでの HTTP セッションの管理」『SQL Anywhere サーバー プログラミング』
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

例

次の例は、sa_set_http_option を使用して、データベースの文字セットエンコードに関する Web サービスの優先度を指定する方法を示します。2 番目の選択肢として UTF-8 エンコードが指定されています。アスタリスク (*) は、クライアントで最も優先される文字セットエンコードが Web サーバーでサポートされている場合に、Web サービスでその文字セットエンコードが使用されることを示しています。

```
CALL sa_set_http_option( 'AcceptCharset', '+,UTF-8,*');
```

次の例は、sa_set_http_option を使用して、Web サービスで使用されている文字エンコードを正しく識別する方法を示します。この例では、Web サーバーは 1251CYR データベースに接続されており、キリル語のアルファベットを含む HTML ドキュメントを任意の Web ブラウザーに提供するように準備されています。

```
CREATE 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();
```

Firefox などの Web ブラウザーから Web サービスに配信される、次の Accept-Charset ヘッダーを例に、使用する正しい文字セットエンコードを確立するプロセスについて説明します。このヘッダーは、ブラウザーでは ISO-8859-1 と UTF-8 のエンコードが優先されるものの、他のエンコードも処理できることを示しています。

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

転送される Web ページにキリル語の文字が含まれているため、Web サービスでは ISO-8859-1 文字セットエンコードは受け入れられません。Web サービスでは、sa_set_http_option の呼び出しで指定されているように、ISO-8859-5 または UTF-8 エンコードが優先されます。この例では、UTF-8 エンコードが選択されます (両方で許容されるため)。データベース接続プロパティ 'CharSet' は、どのエンコードが Web サービスで選択されたかを示します。sa_set_http_header プロシージャーは、Web ブラウザーに対して HTML ドキュメントのエンコードを示すために使用します。

```
Content-Type: text/html; charset=UTF-8
```

Web ブラウザーで Accept-Charset が指定されていない場合、Web サービスではデフォルトで最初の優先エンコード ISO-8859-5 が使用されます。sa_set_http_header プロシージャーは、HTML ドキュメントのエンコードを示すために使用します。

```
Content-Type: text/html; charset=ISO_8859-5
```

次の例では、ユニークな HTTP セッション識別子を設定します。

```
DECLARE sessionid VARCHAR(30);
DECLARE tm TIMESTAMP;
SET tm = NOW(*);
SET sessionid = 'MySessions_' ||
  CONVERT( VARCHAR, SECONDS(tm)*1000 + DATEPART(millisecond,tm));
CALL sa_set_http_option('SessionID', sessionid);
```

次の例では、HTTP セッションのタイムアウトを 5 分に設定します。

```
CALL sa_set_http_option('SessionTimeout', '5');
```

sa_set_soap_header システムプロシージャー

SOAP 応答の SOAP ヘッダーの設定を許可します。このプロシージャーは、SOAP Web サービスから呼び出されたストアドプロシージャー内で使用されます。

構文

sa_set_soap_header(
fldname,

val
)

引数

- ***fldname*** 特定のヘッダーエントリを参照するときに使用するユニークな文字列であるヘッダーキーを指定する VARCHAR パラメーター (*val* の localname と同じである必要はありません)。

- ***val*** SOAP ヘッダー要素の範囲内にあるトップレベルのヘッダーエントリとその子の未加工 XML を指定する VARCHAR パラメーター。

備考

このプロシージャーで設定されたすべての SOAP ヘッダーエントリは、SOAP 応答メッセージを送信するときに、SOAP ヘッダー要素内でシリアル化されます。NULL の *val* はシリアル化されません。SOAP 応答のヘッダーエントリが存在しない場合、SOAP エンベロープ内で内包するヘッダー要素は作成されません。

パーミッション

なし。

関連する動作

なし。

参照

- 「チュートリアル：SQL Anywhere を使用した SOAP/DISH サービスへのアクセス」『SQL Anywhere サーバー プログラミング』
- 「Web サービス関数」147 ページ
- 「Web サービスシステムプロシージャー」1030 ページ

例

次の例は、Hello に対する SOAP ヘッダーのウェルカムメッセージを設定します。

```
CALL sa_set_soap_header( 'welcome', '<welcome>Hello</welcome>' )
```

sa_set_tracing_level システムプロシージャー

診断トレーシングテーブルに格納するトレーシング情報のレベルを初期化します。

構文

sa_set_tracing_level(
level
[, *specified_scope*
, *specified_name*]
[, *do_commit*]
)

引数

- ***level*** 実行する診断トレーシングのレベルを指定する INTEGER パラメーター。使用できる値は、次のとおりです。

  - **0** トレーシングデータを生成しません。このレベルは、トレーシングセッションを開いたままにしますが、トレーシングデータを診断トレーシングテーブルに送信しません。

  - **1** 基本レベルのトレーシングを設定します。

  - **2** 中間レベルのトレーシングを設定します。

  - **3** 高レベルのトレーシングを設定します。

- ***specified_scope*** トレーシングのスコープを指定する任意の LONG VARCHAR パラメーター。たとえば、USER、DATABASE、CONNECTION_NAME、TRIGGER などを指定します。

- ***specified_name*** *specified_scope* に示されたオブジェクトの識別子を指定する任意の LONG VARCHAR パラメーター。

- ***do_commit*** このプロシージャーによって挿入されるローを自動的にコミットするかどうかを指定する任意の TINYINT パラメーター。1 (デフォルト) を指定すると、ローは自動的にコミットされます (この設定をおすすめします)。0 を指定すると、ローは自動的にコミットされません。

備考

このプロシージャーは、sa_diagnostic_tracing_level テーブルのローを置換し、トレーシングレベルとスコープはプロシージャーの呼び出し時に指定した設定に変更されます。

レベル 0 に設定しても、トレーシングセッションは停止しません。その代わり、トレーシングセッションはトレーシングデータベースに所属したままで、トレーシングデータは送信されません。レベルが 0 のときでもトレーシングセッションはアクティブです。

このシステムプロシージャーは、プロファイル対象のデータベースから呼び出す必要があります。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「カスタマイズされた診断トレーシングレベル」『SQL Anywhere サーバー SQL の使用法』
- 「診断トレーシングのスコープ」『SQL Anywhere サーバー SQL の使用法』
- 「sa_diagnostic_tracing_level テーブル」1022 ページ
- 「診断トレーシング」『SQL Anywhere サーバー SQL の使用法』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例は、トレーシングレベルを 1 に設定します。つまり、パフォーマンスカウンターデータや実行される文のサンプル用としてデータベース全体をプロファイルします。

```
CALL sa_set_tracing_level( 1 );
```

次の例は、トレーシングレベルを 3 に設定し、ユーザー AG84756 を指定します。つまり、AG84756 と関連付けられているアクティビティのみがトレースされます。

```
CALL sa_set_tracing_level( 3, 'user', 'AG84756' );
```

sa_snapshots システムプロシージャー

現在アクティブなスナップショットのリストを返します。

構文

sa_snapshots( )

結果セット

カラム名	データ型	説明
connection_num	INT	スナップショットが実行されている接続の接続 ID。
start_sequence_num	UNSIGNED BIGINT	スナップショットを識別するユニークな番号。
statement_level	BIT	スナップショットを statement-snapshot または readonly-statement-snapshot で作成した場合は true。それ以外の場合は false。

備考

文のスナップショットは、1 つの接続に複数存在できます。文のスナップショットアイソレーションレベルで実行されるネストされた文、またはインターリーブされた文の場合、それぞれ最初の読み取りまたは更新で異なる文のスナップショットが開始されます。

トランザクションのスナップショットは、通常は 1 つの接続に 1 つしかありません (statement_level=0 で、1 つの connection in sa_snapshots につき 1 エントリ)。ただし、カーソルに関連付けられたスナップショットは、カーソルの最初のフェッチ後も変化しません。また、WITH HOLD で開いたカーソルは、コミットまたはロールバックの間、開いたままです。カーソルがスナップショットに関連付けられている場合は、そのスナップショットも継続されます。そのため、複数トランザクションのスナップショットが同じ connection_num に対して存在する可能性があります。つまり、1 つは現在のトランザクションのスナップショット、WITH HOLD カーソルのために継続している古いトランザクションのスナップショットには 1 つ以上という場合です。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_transactions システムプロシージャー」1205 ページ
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』

sa_split_list システムプロシージャー

デリミターで区切った値の文字列を使用して、ローのセットを返します (各値に 1 つのロー)。

構文

sa_split_list(
str
[, *delim*]
[, *maxlen*]
)

引数

- ***str*** 分割する値を含む文字列を指定する LONG VARCHAR パラメーター。各値は *delim* で区切られます。
- ***delim*** *str* の値を分割するときに使用するデリミターを指定する任意の CHAR(10) パラメーター。デリミターは、任意の文字の文字列 (10 バイトまで) にできます。*delim* が指定されない場合、デフォルトでカンマが使用されます。
- ***maxlen*** 戻り値の最大長を指定する任意の INTEGER パラメーター。たとえば、*maxlen* が 3 に設定されている場合、結果セットの値は 3 文字の長さにトランケートされます。0 (デフォルト) を指定すると、値は任意の長さで指定できます。

結果セット

カラム名	データ型	説明
line_num	INTEGER	ローのシーケンス番号。
row_value	LONG VARCHAR	文字列の値。必要に応じて、*maxlen* にトランケートされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

sa_split_list プロシージャーは、デリミターで区切られた値のリストが含まれる文字列を受け取り、1 つのローに 1 つの値を入れた結果セットを返します。これは、LIST 関数 [集合] が実行するアクションの逆です。文字列が次の値の場合、row_value には空の文字列が返されます。

- *delim* で始まる
- 文字列の途中で *delim* が 2 つ連続するインスタンスを含む
- *delim* で終わる

入力文字列内の空白スペースには意味があります。デリミターがスペース文字の場合、入力文字列に余分なスペースがあると結果セットの中に余分なローが入ります。デリミターがスペース文字ではない場合、入力文字列のスペースは結果セット内の値から削除されません。

パーミッション

なし。

関連する動作

なし。

参照

- 「LIST 関数 [集合]」268 ページ

例

次のクエリは、色が黒い製品のリストを返します。

```
SELECT list( Name )
  FROM Products
  WHERE Color = 'Black';
```

list (Products.Name)
Tee Shirt,Baseball Cap,Visor,Shorts

次の例では、sa_split_list プロシージャーを使用して、集約されたリストから元の結果セットを返します。

```
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

次の例は、ワードごとに 1 つのローを返します。row_value が空の文字列のローを返さないようにするには、WHERE 句を指定する必要があります。

```
SELECT *
  FROM sa_split_list( 'one||three|four||six|', '|' )
  WHERE row_value <> '';
```

line_num	row_value
1	one
3	three
4	four
6	six

次の例は、ProductsWithColor というプロシージャーが作成されます。呼び出されると、ProductsWithColor プロシージャーは sa_split_list を使用して、ユーザーが指定した色の値を解析し、Products テーブルの Color カラムを検索し、ユーザーが規定した色のいずれかに一致する製品ごとに名前、説明、サイズ、色を返します。

次のプロシージャーを呼び出した結果は、色が白または黒であるすべての製品の名前、説明、サイズ、色です。

```
CREATE 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;
go

SELECT * from ProductsWithColor( 'white,black' );
```

sa_statement_text システムプロシージャー

各行に項目が 1 つずつ表示されるように、SELECT 文をフォーマットします。これは、すべての改行文字が削除されている要求ログから、長い文を表示する場合に便利です。

構文

sa_statement_text(*txt* **)**

引数

- ***txt*** SELECT 文を指定する LONG VARCHAR パラメーター。

備考

txt は文字列として (一重引用符で囲んで)、または文字列式として入力します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_get_request_times システムプロシージャー」1092 ページ
- 「sa_get_request_profile システムプロシージャー」1091 ページ

例

次の呼び出しは、各行に項目が 1 つずつ表示されるように、SELECT 文をフォーマットします。

```
CALL sa_statement_text( 'SELECT * FROM car WHERE name="Audi"' );
```

	stmt_text
1	SELECT *
2	FROM car
3	WHERE name = 'Audi'

sa_table_fragmentation システムプロシージャー

データベーステーブルの断片化に関する情報をレポートします。

構文

sa_table_fragmentation(
  [*tbl_name*
  [, *owner_name*]]
)

引数

- ***tbl_name*** 断片化を調べるテーブルの名前を指定する任意の CHAR(128) パラメーター。
- ***owner_name*** *tbl_name* の所有者を指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
TableName	CHAR(128)	テーブルの名前。
rows	UNSIGNED INTEGER	テーブル内のローの数。

カラム名	データ型	説明
row_segments	UNSIGNED BIGINT	テーブル内のローセグメントの数。
segs_per_row	DOUBLE	ローあたりのセグメントの数。

備考

データベース管理者は、このプロシージャーを使ってデータベースのテーブルの断片化に関する情報を取得できます。引数を指定しないと、データベースにあるすべてのテーブルの結果が返されます。

データベーステーブルの断片化が極端に進んだときは、REORGANIZE TABLE を実行するか、データベースを再構築して、テーブルの断片化を減らし、パフォーマンスを向上させることができます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「テーブルの断片化削減」『SQL Anywhere サーバー SQL の使用法』
- 「データベースの再構築」『SQL Anywhere サーバー SQL の使用法』
- 「REORGANIZE TABLE 文」877 ページ

例

```
CALL sa_table_fragmentation( 'Products','GROUPO' );
```

sa_table_page_usage システムプロシージャー

データベーステーブルのページの使用状況に関する情報をレポートします。

構文

sa_table_page_usage( )

結果セット

カラム名	データ型	説明
TableId	UNSIGNED INTEGER	テーブル ID。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
TablePages	INTEGER	テーブルで使用されるテーブルページの数。
PctUsedT	INTEGER	使用されているテーブルページ領域の割合。
IndexPages	INTEGER	テーブルで使用されるインデックスページの数。
PctUsedI	INTEGER	使用されているインデックスページ領域の割合。
PctOfFile	INTEGER	テーブルが使用しているデータベースファイル総数の割合。
TableName	CHAR(128)	テーブル名。

備考

結果には、情報ユーティリティで提供される情報と同じ内容が含まれています。progress_messages データベースオプションが Raw または Formatted に設定されている場合は、sa_table_page_usage システムプロシージャーの実行中にデータベースサーバーからクライアントに進行メッセージが送信されます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「情報ユーティリティ (dbinfo)」『SQL Anywhere サーバー データベース管理』
- 「progress_messages オプション」『SQL Anywhere サーバー データベース管理』

sa_table_stats システムプロシージャー

各テーブルから読み込まれたページ数に関する情報をレポートします。

構文

sa_table_stats( )

結果セット

カラム名	データ型	説明
table_id	INT	テーブル ID。
creator	CHAR(128)	テーブルの作成者のユーザー名。
table_name	CHAR(128)	テーブル名。
count	UNSIGNED BIGINT	SYSTAB から取得したテーブルの推定ロー数。
table_page_count	UNSIGNED BIGINT	テーブルで使用されるメインページの数。
table_page_cached	UNSIGNED BIGINT	キャッシュに現在格納されているテーブルページの数。
table_page_reads	UNSIGNED BIGINT	メインテーブルのページに実行されるページ読み込み数。
ext_page_count	UNSIGNED BIGINT	テーブルの推定ページ数。
ext_page_cached	UNSIGNED BIGINT	今後の使用のために予約されています。
ext_page_reads	UNSIGNED BIGINT	今後の使用のために予約されています。

備考

sa_table_stats プロシージャーが返す各ローは、ページの統計情報がオプティマイザーによって管理されているテーブルを記述しています。sa_table_stats プロシージャーは、キャッシュメモリを使用しているテーブルや、各テーブルでディスクの読み込みが実行される回数を検索するときに使用されます。たとえば、sa_table_stats プロシージャーを使用して、最も多いディスク読み込みを生成しているテーブルを検索できます。このプロシージャーの結果は推定値であり、診断目的以外には使用しないでください。

table_page_cached カラムは、現在キャッシュに格納されているテーブルのページ数を示します。table_page_reads カラムは、オプティマイザーがテーブルのカウントを管理し始めてからディスクから読み込まれたテーブルのページ数を示します。これらの統計情報は、データベースに永続的には格納されません。これは、テーブルが初めてメモリにロードされた後、テーブルに行われたアクティビティを表します。

パーミッション

DBA 権限。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

関連する動作

なし。

参照

● 「SYSTAB システムビュー」 1311 ページ

sa_text_index_stats システムプロシージャー

データベース内のテキストインデックスに関する統計情報を返します。

構文

sa_text_index_stats( )

備考

sa_text_index_stats システムプロシージャーは、データベース内のテキストインデックスごとに統計情報を表示するときに使用します。次の表に、sa_text_index_stats から返される情報を示します。

カラム名	型	説明
owner_id	UNSIGNED INT	テーブルの所有者の ID
table_id	UNSIGNED INT	テーブルの ID
index_id	UNSIGNED INT	テキストインデックスの ID
text_config_id	UNSIGNED BIGINT	インデックスが参照するテキスト設定オブジェクトの ID
owner_name	CHAR(128)	所有者の名前
table_name	CHAR(128)	テーブルの名前
index_name	CHAR(128)	テキストインデックスの名前
text_config_name	CHAR(128)	テキスト設定オブジェクトの名前
doc_count	UNSIGNED BIGINT	テキストインデックスに含まれるインデックス付けされたカラム値の総数
doc_length	UNSIGNED BIGINT	テキストインデックスに含まれるデータの合計長
pending_length	UNSIGNED BIGINT	保留中の変更の合計長

カラム名	型	説明
deleted_length	UNSIGNED BIGINT	保留中の削除の合計長
last_refresh	TIMESTAMP	最後に行った再表示の日付と時刻

IMMEDIATE REFRESH テキストインデックスの場合、pending_length、deleted_length、last_refresh の値は 0 です。

MANUAL REFRESH テキストインデックスの場合、doc_length、pending_length、deleted_length を使用して、テキストインデックスを再表示するかどうかや、実行する再表示の種類 (再構築か増分か) を決定できます。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「テキストインデックスの概念と参照」『SQL Anywhere サーバー SQL の使用法』
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「sa_refresh_text_indexes システムプロシージャー」1158 ページ
- 「sa_text_index_vocab システムプロシージャー」1202 ページ
- 「SYSTEXTIDX システムビュー」1322 ページ

例

次の文は、データベース内のテキストインデックスごとに統計情報を返します。

```
CALL sa_text_index_stats( );
```

sa_text_index_vocab システムプロシージャー

CHAR テキストインデックスに含まれるすべての単語と、各単語が含まれるインデックス値の合計数のリストを返します。

構文

sa_text_index_vocab(
'*indexname*',
'*tabname*',
['*tabowner*']
)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

引数

- ***indexname*** テキストインデックスの名前を指定する CHAR(128) パラメーター。
- ***tabname*** テキストインデックスを構築するテーブルの名前を指定する CHAR(128) パラメーター。
- ***tabowner*** テーブルの所有者を指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
term	VARCHAR(60)	テキストインデックスに含まれる単語。
freq	BIGINT	単語が含まれるインデックス値の数。

備考

sa_text_index_vocab システムプロシージャーは、テキストインデックスに含まれるすべての単語と、各単語が含まれるインデックス値の合計数を返します (一部のインデックス値に単語が複数回含まれる場合は、出現する総数より小さくなります)。

パーミッション

DBA 権限、またはインデックステーブルの SELECT パーミッションが必要です。

関連する動作

なし。

参照

- 「sa_text_index_vocab_nchar システムプロシージャー」1204 ページ
- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「単語とフレーズの検索」『SQL Anywhere サーバー SQL の使用法』
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「sa_refresh_text_indexes システムプロシージャー」1158 ページ
- 「SYSTEXTIDX システムビュー」1322 ページ

例

次の例は、サンプルデータベース内の Products.Description カラムにテキストインデックス VocabTxtIdx を構築します。その次の文は、sa_text_index_vocab システムプロシージャーを実行し、テキストインデックスに含まれるすべての単語を返します。

```
CREATE TEXT INDEX VocabTxtIdx2 ON Products( Description );
SELECT *
  FROM sa_text_index_vocab( 'VocabTxtIdx2', 'Products', 'GROUPO' );
```

term	freq
Cap	2
Cloth	1
Cotton	2
Crew	1
Hooded	1
neck	2
...	...

sa_text_index_vocab_nchar システムプロシージャー

NCHAR テキストインデックスに含まれるすべての単語と、各単語が含まれるインデックス値の合計数のリストを返します。

構文

sa_text_index_vocab(
'*indexname***',**
'*tabname***',**
['*tabowner***']**
)

引数

- ***indexname*** テキストインデックスの名前を指定する CHAR(128) パラメーター。
- ***tabname*** テキストインデックスを構築するテーブルの名前を指定する CHAR(128) パラメーター。
- ***tabowner*** テーブルの所有者を指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
term	NCHAR(60)	テキストインデックスに含まれる単語。
freq	BIGINT	単語が含まれるインデックス値の数。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

sa_text_index_vocab_nchar システムプロシージャーは、テキストインデックスに含まれるすべての単語と、各単語が含まれるインデックス値の合計数を返します (一部のインデックス値に単語が複数回含まれる場合は、出現する総数より小さくなります)。

パーミッション

DBA 権限、またはインデックステーブルの SELECT パーミッションが必要です。

関連する動作

なし。

参照

- 「sa_text_index_vocab システムプロシージャー」1202 ページ
- 「全文検索」『SQL Anywhere サーバー SQL の使用法』
- 「単語とフレーズの検索」『SQL Anywhere サーバー SQL の使用法』
- 「DROP TEXT INDEX 文」733 ページ
- 「REFRESH TEXT INDEX 文」870 ページ
- 「TRUNCATE TEXT INDEX 文」963 ページ
- 「sa_refresh_text_indexes システムプロシージャー」1158 ページ
- 「SYSTEXTIDX システムビュー」1322 ページ

sa_transactions システムプロシージャー

現在アクティブなトランザクションのリストを返します。

構文

sa_transactions( )

結果セット

カラム名	データ型	説明
connection_num	INT	トランザクションが実行されている接続の接続 ID。
transaction_id	INT	データベースサーバーが追跡している間、トランザクションをユニークに識別する ID。ID は古いトランザクション情報を破棄するときに再利用されます。
start_time	TIMESTAMP	トランザクションの開始時のタイムスタンプ。

カラム名	データ型	説明
start_sequence_num	UNSIGNED BIGINT	トランザクションの開始シーケンス番号。
end_sequence_num	UNSIGNED BIGINT	トランザクションがコミットまたはロールバックされた場合はトランザクションの終了シーケンス番号。それ以外の場合は NULL。
committed	bit	トランザクションのステータス。トランザクションが COMMIT で終了した場合は true、ROLLBACK で終了した場合は false、トランザクションがまだアクティブな場合は NULL。
version_entries	unsigned INT	トランザクションが保存したローバージョンの回数。

備考

このプロシージャーは、現在データベースで実行されているトランザクションに関する情報を提供します。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_snapshots システムプロシージャー」1193 ページ
- 「スナップショットアイソレーション」『SQL Anywhere サーバー SQL の使用法』

sa_unload_cost_model システムプロシージャー

現在のコストモデルを特定のファイルにアンロードします。

構文

sa_unload_cost_model (*file_name* **)**

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

引数

- ***file_name*** データのアンロード先となるファイルの名前を指定する CHAR(256) パラメーター。システムプロシージャーを実行するのはデータベースサーバーであるため、*file_name* はデータベースサーバーコンピューター上のファイルを指定します。また、相対的な *file_name* の場合、データベースサーバーの開始ディレクトリに相対的なファイルを指定します。

備考

オプティマイザーは、コストモデルを使用して、クエリの最適なアクセスプランを決定します。データベースサーバーは、各データベースのコストモデルを管理します。データベースのコストモデルは、ALTER DATABASE 文の CALIBRATE SERVER 句を使用して、任意のタイミングで再調整できます。たとえば、データベースを非標準ハードウェアに移動する場合、コストモデルを再調整できます。

sa_unload_cost_model システムプロシージャーを使用すると、コストモデルを ASCII ファイルに保存できます (*file_name*)。次に、別のデータベースにログインして、sa_load_cost_model システムプロシージャーを使用して、最初のデータベースから 2 番目のデータベースへとコストモデルをロードできます。これによって、2 番目のデータベースを再調整する必要はなくなります。

> **注意**
> sa_unload_cost_model システムプロシージャーは、ファイルに CALIBRATE PARALLEL READ 情報を含めません。

大量に類似ハードウェアのインストールがある場合、sa_unload_cost_model システムプロシージャーを使用すると、繰り返しの時間がかかる再調整動作を省略できます。

パーミッション

DBA 権限。

ファイルを作成するには書き込みのパーミッションが必要です。

関連する動作

なし。

参照

- 「ALTER DATABASE 文」408 ページ
- 「sa_load_cost_model システムプロシージャー」1109 ページ
- 「高度：クエリ最適化」『SQL Anywhere サーバー SQL の使用法』

例

次の例は、costmodel8 というファイルにコストモデルをアンロードします。

```
CALL sa_unload_cost_model( 'costmodel8' );
```

sa_user_defined_counter_add システムプロシージャー

指定した量でユーザー定義カウンターの値を調整します。

構文

sa_user_defined_counter_add (*counter_name*
[, *delta*
[, *apply_to_con*
[, *appy_to_db*
[, *apply_to_server*]]]]
)

引数

- ***counter_name*** 値を変更するユーザー定義カウンターの名前を指定する VARCHAR(128) パラメーター。ユーザー定義カウンター名の例としては、UserDefinedCounterRate01 や UserDefinedCounterRaw01 があります。

- ***delta*** ユーザー定義カウンターを増分または減分する量。デフォルトは 1 です。

- ***apply_to_con*** 現在の接続のカウンター値を調整するかどうかを指定する INT パラメーター。0 は値を調整しないことを意味し、1 は値を調整することを意味します。デフォルトは 1 です。

- ***apply_to_db*** データベースのカウンター値を調整するかどうかを指定する INT パラメーター。0 は値を調整しないことを意味し、1 は値を調整することを意味します。デフォルトは 1 です。

- ***apply_to_server*** データベースサーバーのカウンター値を調整するかどうかを指定する INT パラメーター。0 は値を調整しないことを意味し、1 は値を調整することを意味します。デフォルトは 1 です。

備考

このプロシージャーは、*delta* が定義された場合は 1 を返し、それ以外は 0 を返します。*delta* が定義されない場合は 0 を返し、エラーが発生した場合はエラーコードを返します。次のようなエラーがあります。

- カウンター名が無効である。
- apply_to_server、apply_to_db、または apply_to_con parameter の値が無効である。

カウンターへの同時アクセスは自動的に適用されるので、カウンター値は複数の同時要求によって増分される可能性があります。

パーミッション

DBA 権限。

関連する動作

なし。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「sa_user_defined_counter_set システムプロシージャー」1209 ページ
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「ユーザー定義統計情報」『SQL Anywhere サーバー SQL の使用法』
- 「ユーザー定義プロパティ」『SQL Anywhere サーバー データベース管理』

例

次の文は、現在の接続、データベース、データベースサーバーの UserDefinedCounterRate01 の値を 0 に変更します。

```
SELECT dbo.sa_user_defined_counter_set( 'UserDefinedCounterRate01', 0, 1, 1, 1 );
```

sa_user_defined_counter_set システムプロシージャー

指定した値にユーザー定義カウンターを設定します。

構文

sa_user_defined_counter_set (*counter_name*
[, *value*
[, *apply_to_con*
[, *appy_to_db*
[, *apply_to_server*]]]]
)

引数

- ***counter_name*** 値を変更するユーザー定義カウンターの名前を指定する VARCHAR(128) パラメーター。ユーザー定義カウンター名の例としては、UserDefinedCounterRate01 や UserDefinedCounterRaw01 があります。

- ***value*** ユーザー定義カウンターを変更する量。デフォルトは 1 です。

- ***apply_to_con*** 現在の接続のカウンター値を調整するかどうかを指定する INT パラメーター。0 は値を調整しないことを意味し、1 は値を調整することを意味します。デフォルトは 1 です。

- ***apply_to_db*** データベースのカウンター値を調整するかどうかを指定する INT パラメーター。0 は値を調整しないことを意味し、1 は値を調整することを意味します。デフォルトは 1 です。

- ***apply_to_server*** データベースサーバーのカウンター値を調整するかどうかを指定する INT パラメーター。0 は値を調整しないことを意味し、1 は値を調整することを意味します。デフォルトは 1 です。

備考

このプロシージャーは、*value* が定義された場合は 1 を返し、それ以外は 0 を返します。*value* が定義されない場合は 0 を返し、エラーが発生した場合はエラーコードを返します。次のようなエラーがあります。

- カウンター名が無効である。
- apply_to_server、apply_to_db、または apply_to_con parameter の値が無効である。

カウンターへの同時アクセスは自動的に適用されるので、カウンター値は複数の同時要求によってリセットされる可能性があります。

パーミッション

DBA 権限。

関連する動作

なし。

参照

- 「sa_user_defined_counter_add システムプロシージャー」1208 ページ
- 「接続プロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「ユーザー定義プロパティ」『SQL Anywhere サーバー データベース管理』
- 「ユーザー定義統計情報」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、現在の接続、データベース、データベースサーバーの UserDefinedCounterRate01 の値を 0 に設定します。

```
SELECT dbo.sa_user_defined_counter_set( 'UserDefinedCounterRate01', 0, 1, 1, 1 );
```

sa_validate システムプロシージャー

データベースの全部または一部を検証します。

構文

sa_validate(
[*tbl_name* [, *owner_name*]]
)

引数

- ***tbl_name*** 検証するテーブルまたはマテリアライズドビューの名前を指定する任意の VARCHAR(128) パラメーター。

- ***owner_name*** 所有者を指定する任意の VARCHAR(128) パラメーター。所有者のみを指定すると、この所有者が所有するすべてのテーブルとマテリアライズドビューが検証されます。

パーミッション

DBA 権限。

関連する動作

なし。

備考

指定される引数	検証タイプ
なし。	データベース内のすべてのテーブル、マテリアライズドビュー、インデックスが検証されます。チェックサムの検証を含め、データベース自体も検証されます。
owner_name	指定されたユーザーが所有するすべてのテーブル、マテリアライズドビュー、インデックスが検証されます。
tbl_name	現在のユーザーが所有している指定されたテーブルまたはマテリアライズドビュー、およびそのすべてのインデックスが検証されます。
owner_name と *tbl_name*	指定されたユーザーが所有している指定されたテーブルまたはマテリアライズドビュー、およびそのすべてのインデックスが検証されます。

tbl_name と *owner_name* の値は文字列であり、引用符で囲む必要があります。

プロシージャーは Messages という 1 つのカラムを返します。検証時にエラーが返されると、このカラムに表示されます。エラーなく検証が成功した場合は、カラムに「**エラーは見つかりませんでした。**」が格納されます。

警告
テーブルまたはデータベース全体の検証は、データベースに変更を加えている接続がない場合に実行してください。そうしないと、実際に破損していなくても、何らかの形でデータベースが破損したことを示すエラーがレポートされます。

例

次の文は、DBA が所有するテーブルとマテリアライズドビューの検証を実行します。

```
CALL sa_validate( owner_name = 'DBA' );
```

sa_verify_password システムプロシージャー

現在のユーザーのパスワードを検証します。

構文

sa_verify_password(*curr_pswd* **)**

引数

- ***curr_pwsd*** 現在のデータベースユーザーのパスワードを指定する CHAR(128) パラメーター。

備考

このプロシージャーは sp_password で使用されます。パスワードが一致した場合、そのパスワードは受け入れられます。パスワードが一致しなかった場合、エラーが返されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「Adaptive Server Enterprise システムプロシージャー」 1033 ページ

sp_forward_to_remote_server システムプロシージャー

構文

sp_forward_to_remote_server (
server_name,
sql
)

引数

- ***server_name*** SQL 文が実行されるリモートサーバーの名前を指定する CHAR(128) パラメーター。
- ***sql*** リモートサーバーで実行する SQL 文を指定する LONG VARCHAR パラメーター。

備考

このプロシージャーでは、アプリケーションがリモートサーバー上で SQL 文を実行し、その文によって生成される結果セットを取得できます。SQL 文はリモートサーバーにそのまま送信されるため、SQL Anywhere が文を解析する必要はありません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

このシステムプロシージャーを使用するには、CREATE SERVER 文でリモートサーバーを定義する必要があります。

FORWARD TO 文とは異なり、sp_forward_to_remote_server はプロシージャー内で使用できます。ただし、このストアドプロシージャーは、リモート結果セットのスキーマが任意なので、SELECT 文の FROM 句の中では使用できません。リモート結果セットのフェッチは、sp_forward_to_remote_server プロシージャーで呼び出されるストアドプロシージャーでカーソルを宣言することで実行できます。

注意
SQL 文が複数の結果セットを返す場合、sp_forward_to_remote_server ストアドプロシージャーは順に各リモート結果セットを返します。

パーミッション

なし。

関連する動作

このストアドプロシージャーの実行に、ローカルで関連する動作はありません。しかし、リモートサーバー上で実行される SQL 文は任意なので、リモートサーバーでは関連する動作がある可能性があります。

参照

- 「FORWARD TO 文」757 ページ
- 「CREATE SERVER 文」618 ページ

例

次の例は sp_forward_to_remote_server ストアドプロシージャーとローカル関数を使用して、リモートの SQL Anywhere 12 データベースが所有しているテーブル数を判断します。

1. 次のコマンドを実行して、*testdb1* と *testdb2* という名前の 2 つのデータベースを作成します。

```
dbinit testdb1
dbinit testdb2
```

2. 両方のデータベースを同じ SQL Anywhere サーバーで起動します。

```
dbeng12 -n mytest testdb1 testdb2
```

3. Interactive SQL から *testdb1* に接続します。

```
dbisql -c "uid=DBA;pwd=sql;server=mytest;dbn=testdb1;links=SharedMemory"
```

4. *testdb2* にリモートデータアクセスサーバーを作成します。

```
CREATE SERVER rem CLASS 'saodbc' USING 'driver=SQL Anywhere 12 ;
  eng = mytest ; dbn = testdb2 ; links = SharedMemory'
```

5. 外部ログインは必要ないことを前提とします。sp_forward_to_remote_server を呼び出すローカル関数を作成します。

```
CREATE FUNCTION fetch_num_remote_tables ( IN server char(128) ) RETURNS int
   BEGIN
      DECLARE num_tables int;
      DECLARE curs CURSOR FOR CALL sp_forward_to_remote_server
        ( server, 'SELECT COUNT(*) FROM sys.systable' );
      OPEN curs;
      FETCH next curs INTO num_tables;
      CLOSE curs;
      RETURN num_tables;
   END
```

6. リモートサーバーからテーブルの数をフェッチします。

```
SELECT fetch_num_remote_tables( 'rem' )
```

sp_get_last_synchronize_result システムプロシージャー

SYNCHRONIZE 文によって最後に開始された同期に関する情報を返します。

構文

sp_get_last_synchronize_result (
conn_id,
complete_only
)

引数

- ***conn_id*** SYNCHRONIZE 文が実行された接続の接続 ID 番号を指定する INTEGER パラメーター。値を指定しない場合、現在の接続の接続 ID が使用されます。

- ***complete_only*** 完了した同期に関する情報が返されるようにするには、この BIT パラメーターを 1 に設定します。現在アクティブな同期に関する情報が返されるようにするには、このパラメーターを 0 に設定します。

結果セット

カラム名	データ型	説明
row_id	BIGINT	ローがテーブルに挿入された順序を判断するために使用するテーブルのプライマリキー。
conn_id	UNSIGNED INT	このイベントを生成した SYNCHRONIZE 文を実行した接続の接続 ID 番号。
result_time	TIMESTAMP	イベントが synchronize_results テーブルに追加された時刻。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
result_type	CHAR(128)	イベントのタイプ。
parm_id	INTEGER	各イベントには対応する 0 個以上のパラメーターが存在する可能性があります。parm_id カラムは、各イベントに対応するパラメーターの順序を決定します。
parm_result	long varchar	イベントパラメーターに関連付けられているメッセージテキスト。

備考

過去または現在の同期の詳細を表示するには、グローバル共有テンポラリテーブルの synchronize_results と synchronize_parameters を直接問い合わせる代わりに、sp_get_last_synchronize_result ストアドプロシージャーを使用できます。このストアドプロシージャーでは、指定した接続 ID 番号に対する最後の同期の結果のみを返します。パラメーターを何も指定しないと、現在の接続に対して完了した最後の同期が返されます。

また、このストアドプロシージャーを使用して、現在の接続以外の接続に対する同期の進行状況をモニターすることもできます。別の接続に対する同期の進行状況をモニターするには、次の手順に従います。

1. SELECT CONNECTION_PROPERTY 文を実行して、現在の接続の接続 ID を確認します。
2. SELECT CONNECTION_PROPERTY 文によって返された接続 ID を使用して SYNCHRONIZE 文を実行します。
3. 別の接続に対して、SELECT CONNECTION_PROPERTY 文を実行し、*@complete_only* パラメーターを 0 に設定します。指定した接続に対する最後の同期に関する情報が返されます (同期が未完了の場合も同様)。

synchronize_parameters テーブルのイベントとそれに関連付けられた parm_id 値のリストを次に示します。

イベント	parm_id 値	説明
DBSC_EVENTTYPE_ERROR_MSG	0	エラーメッセージ文。
	1	メッセージに関連したメッセージ ID。
DBSC_EVENTTYPE_WARNING_MSG	0	警告メッセージのテキスト。

イベント	parm_id 値	説明
	1	メッセージに関連したメッセージ ID。
DBSC_EVENTTYPE_INFO_MSG	0	情報メッセージのテキスト。
	1	メッセージに関連したメッセージ ID。
DBSC_EVENTTYPE_PROGRESS_INDEX	0	新しい進行状況のインデックス値。
DBSC_EVENTTYPE_PROGRESS_TEXT	0	新しい進行状況のテキスト文。
DBSC_EVENTTYPE_TITLE	0	新しいウィンドウタイトル。
DBSC_EVENTTYPE_SYNC_DONE	0	同期からの exit コード。0 は成功。
DBSC_EVENTTYPE_ML_CONNECT	0	使用されている通信プロトコル。
	1	使用されているネットワークプロトコル。
DBSC_EVENTTYPE_DOWNLOAD_COMMITTED	0	コミットされた insert/update 操作の数。
	1	コミットされた delete 操作の数。
DBSC_EVENTTYPE_UPLOAD_SENT	0	アップロードされた insert 操作の数。
	1	アップロードされた update 操作の数。
	2	アップロードされた delete 操作の数。

パーミッション

DBA

関連する動作

なし。

参照

- 「SYNCHRONIZE 文 [Mobile Link]」952 ページ
- DBSC_Event 構造体 [Dbmlsync .NET]『Mobile Link クライアント管理』

例

次の例では、現在の接続に対して完了した最後の同期に関する情報を返します。

```
CALL sp_get_last_synchronize_result();
```

次の例では、sp_get_last_synchronize_result システムプロシージャーを呼び出すときに名前付きパラメーターを使用し、接続 ID 25 から開始された最後に完了した同期についての情報を返します。

```
CALL sp_get_last_synchronize_result(
    @conn_id=25,
    @complete_only=1);
```

sp_login_environment システムプロシージャー

ユーザーがログインするときの接続オプションを設定します。

構文

sp_login_environment( )

備考

sp_login_environment は、デフォルトで login_procedure データベースオプションによって呼び出されるプロシージャーです。

このプロシージャーを編集しないことをおすすめします。ログイン環境を変更するには、異なるプロシージャーを指すよう login_procedure オプションを変更します。

sp_login_environment プロシージャーのテキストを次に示します。

```
CREATE PROCEDURE dbo.sp_login_environment( )
BEGIN
  IF connection_property( 'CommProtocol' ) = 'TDS' THEN
    CALL dbo.sp_tsql_environment( )
  END IF
END;
```

パーミッション

なし。

関連する動作

なし。

参照

- 「login_procedure オプション」『SQL Anywhere サーバー データベース管理』

sp_remote_columns システムプロシージャー

リモートテーブルにあるカラムとそれらのデータ型の記述のリストを生成します。

このシステムプロシージャーを使用するには、サーバーを CREATE SERVER 文で定義します。

構文

sp_remote_columns(
server_name,
table_name
[, *table_owner*
[, *table_qualifier*]]
)

引数

- ***server_name*** CREATE SERVER 文で指定されたサーバー名を含む文字列を指定する CHAR(128) パラメーター。
- ***table_name*** リモートテーブルの名前を指定する CHAR(128) パラメーター。
- ***table_owner*** *@table_name* の所有者を指定する任意の CHAR(128) パラメーター。
- ***table_qualifier*** 任意の CHAR(128) パラメーターを使用して、*@table_name* が格納されているデータベースの名前を指定します。

結果セット

カラム名	データ型	説明
database	CHAR(128)	データベース名。
owner	CHAR(128)	データベース所有者名。
table_name	CHAR(128)	テーブル名。
column_name	CHAR(128)	カラムの名前。
domain_id	SMALLINT	カラムのデータ型を示す INTEGER。
width	INTEGER	このカラムの意味は、データ型によって異なります。character 型の場合、width は文字数を表します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
scale	SMALLINT	このカラムの意味は、データ型によって異なります。NUMERIC データ型の場合、scale とは小数点以下の桁数です。
nullable	SMALLINT	NULL カラム値が許可される場合、値は 1 です。それ以外の場合、値は 0 です。
base_type_str	CHAR(4096)	カラムの物理的な型を表す注釈付きの型文字列。

備考

CREATE EXISTING TABLE 文を入力していて、カラムリストを指定している場合は、リモートテーブルで使用可能なカラムのリストを取得しておくと役立つ場合があります。sp_remote_columns はリモートテーブルのカラムとそのデータ型の記述のリストを生成します。データベースを指定する場合は、所有者または値 NULL を指定してください。

標準と互換性

- **Sybase** Open Client/Open Server でサポートされています。

パーミッション

なし。

関連する動作

なし。

参照

- 「リモートデータアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「リモートデータアクセスのサーバークラス」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE SERVER 文」618 ページ

例

次の例は、Adaptive Server Enterprise サーバー asetest 上にある運用データベースの SYSOBJECTS テーブルからカラムを返します。所有者は指定されていません。

```
CALL sp_remote_columns( 'asetest', 'sysobjects', null, 'production' );
```

sp_remote_exported_keys システムプロシージャー

指定されたプライマリテーブルに外部キーを持つテーブルに関する情報を表示します。

このシステムプロシージャーを使用するには、サーバーを CREATE SERVER 文で定義します。

構文

sp_remote_exported_keys(
server_name
, *sp_name*
[, *sp_owner*
[, *sp_qualifier*]]
)

引数

- ***server_name*** プライマリテーブルが格納されているサーバーを指定する CHAR(128) パラメーター。
- ***sp_name*** プライマリキーを格納するテーブルを指定する CHAR(128) パラメーター。
- ***sp_owner*** プライマリテーブルの所有者を指定する任意の CHAR(128) パラメーター。
- ***sp_qualifier*** プライマリテーブルを格納するデータベースを指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
pk_database	CHAR(128)	プライマリキーテーブルがあるデータベース。
pk_owner	CHAR(128)	プライマリキーテーブルの所有者。
pk_table	CHAR(128)	プライマリキーテーブル。
pk_column	CHAR(128)	プライマリキーカラムの名前。
fk_database	CHAR(128)	外部キーテーブルがあるデータベース。
fk_owner	CHAR(128)	外部キーテーブルの所有者。
fk_table	CHAR(128)	外部キーテーブル。
fk_column	CHAR(128)	外部キーカラムの名前。
key_seq	SMALLINT	キーのシーケンス番号。
fk_name	CHAR(128)	外部キーの名前。
pk_name	CHAR(128)	プライマリキーの名前。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このプロシージャーは、特定のプライマリテーブルに外部キーを持つリモートテーブルに関する情報を提供します。sp_remote_exported_keys システムプロシージャーの結果セットには、データベース、所有者、テーブル、カラム、プライマリキーと外部キーの両方の名前、外部キーカラムの外部キーシーケンスが含まれます。基本となる ODBC と JDBC 呼び出しのために結果セットが変わる場合もありますが、外部キーのテーブルとカラムに関する情報は常に返されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「CREATE SERVER 文」618 ページ
- 「外部キー」『SQL Anywhere サーバー SQL の使用法』

例

この例では、システムプロシージャーを呼び出すときに名前付きパラメーターを使用し、サーバー asetest 上の運用データベースの SYSOBJECTS テーブルに外部キーを持つリモートテーブルに関する情報を返します。

```
CALL sp_remote_exported_keys(
    @server_name='asetest',
    @sp_name='sysobjects',
    @sp_qualifier='production' );
```

sp_remote_imported_keys システムプロシージャー

指定された外部キーに対応するプライマリキーを持つリモートテーブルに関する情報を提供します。

このシステムプロシージャーを使用するには、サーバーを CREATE SERVER 文で定義します。

構文

sp_remote_imported_keys(
server_name
, *sp_name*
[, *sp_owner*
[, *sp_qualifier*]]
)

引数

- ***server_name*** 外部キーテーブルが配置されているサーバーを指定する任意の CHAR(128) パラメーター。このパラメーターの値は必須です。

- ***sp_name*** 外部キーを格納するテーブルを指定する任意の CHAR(128) パラメーター。このパラメーターの値は必須です。
- ***sp_owner*** 外部キーテーブルの所有者を指定する任意の CHAR(128) パラメーター。
- ***sp_qualifier*** 外部キーテーブルを含むデータベースを指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
pk_database	CHAR(128)	プライマリキーテーブルがあるデータベース。
pk_owner	CHAR(128)	プライマリキーテーブルの所有者。
pk_table	CHAR(128)	プライマリキーテーブル。
pk_column	CHAR(128)	プライマリキーカラムの名前。
fk_database	CHAR(128)	外部キーテーブルがあるデータベース。
fk_owner	CHAR(128)	外部キーテーブルの所有者。
fk_table	CHAR(128)	外部キーテーブル。
fk_column	CHAR(128)	外部キーカラムの名前。
key_seq	SMALLINT	キーのシーケンス番号。
fk_name	CHAR(128)	外部キーの名前。
pk_name	CHAR(128)	プライマリキーの名前。

備考

外部キーは、対応するプライマリキーを持つ別のテーブル内のローを参照します。このプロシージャーを使用すると、特定の外部テーブルに対応するプライマリキーを持つリモートテーブルのリストを取得できます。sp_remote_imported_keys の結果セットには、データベース、所有者、テーブル、カラム、プライマリキーと外部キーの両方の名前、外部キーカラムの外部キーシーケンスが含まれます。基本となる ODBC と JDBC 呼び出しのために結果セットが変わる場合もありますが、プライマリキーのテーブルとカラムに関する情報は常に返されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「CREATE SERVER 文」618 ページ
- 「外部キー」『SQL Anywhere サーバー SQL の使用法』

例

次の例では、プロシージャーを呼び出すときに名前付きパラメーターを使用し、asetest サーバーの SYSOBJECTS テーブルの外部キーに対応するプライマリキーを持つテーブルを返します。

```
CALL sp_remote_imported_keys(
    @server_name='asetest',
    @sp_name='sysobjects',
    @sp_qualifier='production' );
```

sp_remote_primary_keys システムプロシージャー

リモートデータアクセスを使用してリモートテーブルについてのプライマリキー情報を提供します。

構文

sp_remote_primary_keys(
server_name
, *table_name*
[**,** *table_owner*
[**,** *table_qualifier*]]
)

引数

- ***server_name*** リモートテーブルが配置されているサーバーを指定する CHAR(128) パラメーター。
- ***table_name*** リモートテーブルを指定する CHAR(128) パラメーター。
- ***table_owner*** リモートテーブルの所有者を指定する任意の CHAR(128) パラメーター。
- ***table_qualifier*** リモートデータベースの名前を指定する任意の CHAR(128) パラメーター。

結果セット

カラム名	データ型	説明
database	CHAR(128)	リモートデータベースの名前。
owner	CHAR(128)	リモートテーブルの所有者。

カラム名	データ型	説明
table-name	CHAR(128)	リモートテーブル。
column-name	CHAR(128)	カラムの名前。
key-seq	SMALLINT	プライマリキーのシーケンス番号。
pk_name	CHAR(128)	プライマリキーの名前。

備考

このシステムプロシージャーは、リモートデータアクセスを使用してリモートテーブルについてのプライマリキー情報を提供します。

基本となる ODBC/JDBC 呼び出しに違いがあるため、返される情報は、サーバーに指定されたリモートデータアクセスクラスによって、カタログの値やデータベースの値と多少異なります。

標準と互換性

- **Sybase** Open Client/Open Server でサポートされています。

パーミッション

なし。

関連する動作

なし。

sp_remote_tables システムプロシージャー

サーバー上のテーブルのリストを返します。

このシステムプロシージャーを使用するには、サーバーを CREATE SERVER 文で定義します。

構文

sp_remote_tables(
server_name
[, *table_name*
[, *table_owner*
[, *table_qualifier*
[, *with_table_type*]]]]
)

引数

- ***server_name*** リモートテーブルが配置されているサーバーを指定する CHAR(128) パラメーター。

- ***table_name*** リモートテーブルを指定する任意の CHAR(128) パラメーター。
- ***table_owner*** リモートテーブルの所有者を指定する任意の CHAR(128) パラメーター。
- ***table_qualifier*** *table_name* が格納されているデータベースを指定する任意の CHAR(128) パラメーター。
- ***with_table_type*** リモートテーブルのタイプを指定する任意の BIT パラメーター。この引数は、Bit データ型で、0 (デフォルト) と 1 の 2 つの値を指定できます。テーブルのタイプをリストするカラムを結果セットに入れる場合は、値 1 を指定してください。

結果セット

カラム名	データ型	説明
database	CHAR(128)	リモートデータベースの名前。
owner	CHAR(128)	リモートデータベース所有者の名前。
table-name	CHAR(128)	リモートテーブル。
table-type	CHAR(128)	テーブルタイプを指定します。値は、リモートサーバーのタイプによって異なります。たとえば、指定できる値として TABLE、VIEW、SYS、GBL TEMP があります。

備考

データベースサーバーを設定するときに、特定のサーバー上で使用可能なリモートテーブルのリストを取得しておくと役立つ場合があります。このプロシージャーは、サーバー上のテーブルのリストを返します。

このプロシージャーには 5 つのパラメーターを指定できます。テーブル、所有者、またはデータベース名を指定すると、テーブルのリストはその引数に当てはまるものだけに限定されます。

標準と互換性

- **Sybase** Open Client/Open Server でサポートされています。

パーミッション

なし。

関連する動作

なし。

参照

- 「リモートデータアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「リモートデータアクセスのサーバークラス」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE SERVER 文」618 ページ

例

サーバー excel で参照されている ODBC データソースから、使用可能なすべての Microsoft Excel ワークシートのリストを取得します。

```
CALL sp_remote_tables( 'excel' );
```

Adaptive Server Enterprise サーバー asetest の運用データベースにある fred が所有するすべてのテーブルのリストを取得します。

```
CALL sp_remote_tables( 'asetest', null, 'fred', 'production' );
```

sp_servercaps システムプロシージャー

リモートサーバーの機能についての情報を表示します。

このシステムプロシージャーを使用するには、サーバーを CREATE SERVER 文で定義します。

構文

sp_servercaps(*sname* **)**

引数

- ***sname*** CREATE SERVER 文で定義されるサーバーを指定する CHAR(64) パラメーター。*sname* は、CREATE SERVER 文で使用されるサーバー名と同じである必要があります。

備考

このプロシージャーは、リモートサーバーの機能についての情報を表示します。この機能の情報は、どのくらいの SQL 文をリモートサーバーに転送できるかを判断するために使用されます。サーバーの機能をリストする ISYSCAPABILITY システムテーブルは、最初のリモートサーバーに接続するまでは設定されません。

標準と互換性

- **Sybase** Open Client/Open Server でサポートされています。

パーミッション

なし。

関連する動作

なし。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SYSCAPABILITY システムビュー」1255 ページ
- 「SYSCAPABILITYNAME システムビュー」1256 ページ
- 「リモートデータアクセス」『SQL Anywhere サーバー SQL の使用法』
- 「リモートデータアクセスのサーバークラス」『SQL Anywhere サーバー SQL の使用法』
- 「CREATE SERVER 文」618 ページ

例

リモートサーバー testasa に関する情報を表示します。

```
CALL sp_servercaps( 'testasa' );
```

sp_tsql_environment システムプロシージャー

ユーザーが jConnect または Open Client アプリケーションから接続するときの接続オプションを設定します。

構文

sp_tsql_environment( )

備考

sp_login_environment プロシージャーは、login_procedure データベースオプションによって指定されるデフォルトのプロシージャーです。新規接続ごとに、login_procedure で指定されたプロシージャーが呼び出されます。接続に TDS 通信プロトコルを使用する場合 (つまり、Open Client または jConnect 接続の場合)、今度は sp_login_environment が sp_tsql_environment を呼び出します。

このプロシージャーは、デフォルトの Adaptive Server Enterprise の動作との互換性を持つように、データベースオプションを設定します。

デフォルトの動作を変更したい場合は、新しいプロシージャーを作成して、その新しいプロシージャーを指すように login_procedure オプションを変更することをおすすめします。

パーミッション

なし。

関連する動作

なし。

参照

- 「sp_login_environment システムプロシージャー」1217 ページ
- 「login_procedure オプション」『SQL Anywhere サーバー データベース管理』

例

sp_tsql_environment プロシージャーのテキストを次に示します。

```
CREATE PROCEDURE dbo.sp_tsql_environment()
BEGIN
   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';
END
```

st_geometry_dump システムプロシージャー

ジオメトリを最低レベルのコンポーネントジオメトリに分解します。

構文

st_geometry_dump (*geometry* **[,** *options* **])**

引数

- ***geometry*** 分解するジオメトリ値。
- ***options*** セミコロンで区切ったパラメーターと値の VARCHAR(255) 文字列。プロシージャーの出力を設定するために使用できます。

  次の表は、指定できるパラメーターのリストです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター	デフォルト値	指定可能な値	説明
Format	Original	Original、Internal、または Mixed	ジオメトリを返すフォーマット。Original を指定すると、ジオメトリは元のフォーマットで返されます。Internal を指定すると、ジオメトリは正規化フォーマットで返されます。Mixed を指定すると、使用可能な格納フォーマットがどのようなものであっても、1 つのフォーマットにつき 1 ローが返されます。
ExpandPoints	Yes	Yes、No	デフォルトでは、ポイント (ST_LineString や ST_MultiPoint など) を含むジオメトリを分解すると、st_geometry_dump システムプロシージャーは構成ポイントを別々のローに出力します。これらの余分なローが生成されないようにするには、ExpandPoints を No に設定します。

パラメーター	デフォルト値	指定可能な値	説明
MaxDepth	-1	-1、0 以上の任意の数値	デフォルトでは、st_geometry_dump システムプロシージャーは、リーフオブジェクトに達するまで、オブジェクト階層の分解を続けます。MaxDepth パラメーターを設定して、階層内でのジオメトリの分解レベル数を制限できます。値を 0 に設定すると、ルートジオメトリのみが返されます。値を 1 に設定すると、ジオメトリとその直近の子が返されます。それ以降についても同様です。
SetGeom	Yes	Yes、No	st_geometry_dump システムプロシージャーは、元の型階層内のオブジェクトに関連付けられている ST_Geometry のカラムを返します。このカラムが必要でない場合は、SetGeom パラメーターを No に設定して、プロシージャーの実行時間と出力サイズを減少させることができます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター	デフォルト値	指定可能な値	説明
Validate	Basic	None、Basic、Full	デフォルトでは、st_geometry_dump システムプロシージャーは、ジオメトリのロード時にデータベースサーバーで使用される検証ルールを適用し、ロー内のオブジェクトがこれらのルールに一致する場合は、結果セットの Valid カラムを 1 に設定します。Validate パラメーターを None に設定すると、このチェックが無効になり、Full に設定すると、ST_IsValid メソッドによって実行される追加のチェックも適用されます。Full でのチェックは実行に時間がかかります。

戻り値

次の表は、st_geometry_dump プロシージャーによって返される結果を示します。

カラム	データ型	説明
id	UNSIGNED BIGINT	結果内のこのローのユニークな ID。
parent_id	UNSIGNED BIGINT	このオブジェクトの直近の親の ID。
depth	INT	ルートオブジェクトから、このローに関連付けられているオブジェクトまでの深さ。
format	VARCHAR(128)	ジオメトリが元の表現 (Original) であるか、または正規化表現 (Internal) であるか。

カラム	データ型	説明
valid	BIT	ジオメトリが有効 (1) かどうか (Validate オプションで指定したチェックレベルに基づく)。
geom_type	VARCHAR(128)	ジオメトリの型 (ST_GeometryType によって返される)。
geom	ST_Geometry	ジオメトリの仕様。SetGeom パラメーターを No に設定すると、結果セットにジオメトリの仕様は返されません。
xmin	DOUBLE	ジオメトリの x の最小値。
xmax	DOUBLE	ジオメトリの x の最大値。
ymin	DOUBLE	ジオメトリの y の最小値。
ymax	DOUBLE	ジオメトリの y の最大値。
zmin	DOUBLE	ジオメトリの z の最小値。
zmax	DOUBLE	ジオメトリの z の最大値。
mmin	DOUBLE	ジオメトリの m の最小値。
mmax	DOUBLE	ジオメトリの m の最大値。
details	LONG VARCHAR	ジオメトリに関する追加の詳細 (オブジェクトが有効でない理由に関する追加情報など)。

備考

st_geometry_dump システムプロシージャーは、ジオメトリ階層を分解するときに、階層内のオブジェクト (ルートオブジェクトを含む) ごとに 1 つのローを使用します。階層内の各ジオメトリを検証して、各ジオメトリが有効かどうか、有効でない場合はその理由を調査できます。

st_geometry_dump システムプロシージャーの一部の機能は、ST_GeometryN や ST_PointN などの型固有のメソッドを使用して照合できます。

st_geometry_dump システムプロシージャーを使用して、無効なジオメトリを修正できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

なし。

関連する動作

なし。

参照

- ST_Geometry タイプの ST_IsValid メソッド『SQL Anywhere サーバー 空間データサポート』
- 「CREATE SPATIAL REFERENCE SYSTEM 文」635 ページ
- 「st_geometry_on_invalid オプション」『SQL Anywhere サーバー データベース管理』
- STORAGE FORMAT 句、CREATE SPATIAL REFERENCE SYSTEM 文 640 ページ

例

次の例では、多角形 'Polygon ((0 0, 3 0, 3 3, 0 3, 0 0))' をそのコンポーネントジオメトリに分解します。

```
SELECT * FROM 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	...

id	parent_id	depth	format	valid	geom_type	geom	xmin	xmax	ymin	ymax	...
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	...

次の例は、st_geometry_dump システムプロシージャーを使用して、ジオメトリ内の無効なポイントを見つける方法を示します。この例では、LineString に経度 1200 のポイントが含まれています。このため、結果ではそのポイントと LineString の両方が無効 (valid=0) であるとレポートされています。

```
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).

id	parent_id	depth	format	valid	geom_type	geom	xmin	xmax	ymin	ymax	...	details
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	...	

無効なデータが識別されたら、st_geometry_dump システムプロシージャーを他の空間メソッドとともに使用して、無効な要素を修正して有効なジオメトリを組み立てることができます。次の例は、経度 1200 の無効なポイントを経度 120.0 に修正する方法を示します。

```
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';
```

st_geometry_load_shapefile システムプロシージャー

テーブルを作成し、そこに ESRI シェイプファイルをロードします。

構文

st_geometry_load_shapefile (*shp_filename*
, *srid*
, *table_name*
[**,** *table_owner*]
[**,** *shp_encoding*] **)**

引数

- ***shp_filename*** ESRI シェイプファイルのロケーションを指定する VARCHAR(512) パラメーター。ファイル名の拡張子は *.shp* にする必要があります。また、同じディレクトリにある同じベース名の *.shx* ファイルと *.dbf* ファイルを関連付ける必要があります。パスは、クライアントアプリケーションではなく、データベースサーバーを基準にした相対パスを指定します。

- ***srid*** シェイプファイルからのデータを関連付けるための SRID を指定する INTEGER パラメーター。ジオメトリを意味のあるものにするため、指定する SRID は、シェイプファイルで使用されている SRID と同一ではなくても、ジオメトリに対して適切なものにする必要があります。

- ***table_name*** シェイプファイルデータを保持するために作成するテーブル名を指定する VARCHAR(128) カラム。

- ***table_owner*** 作成された新しいテーブルの所有者を指定する任意の VARCHAR(128) カラム。デフォルトの所有者は現在のユーザーです。

- ***shp_encoding*** シェイプファイルを読み込むときに使用するエンコードを指定する任意の VARCHAR(50) パラメーター。デフォルトのエンコードは ISO-8859-1 です。

備考

st_geometry_load_shapefile システムプロシージャーは、最初に、ESRI シェイプファイルで見つかったカラム情報 (名前とデータ型) を使ってテーブル "table_owner"."table_name" を作成します。その後、シェイプファイルからのデータを新しいテーブルにロードします。

このプロシージャーは、sa_describe_shapefile システムプロシージャーと LOAD TABLE...FORMAT SHAPEFILE 文を利用します。

パーミッション

RESOURCE 権限が必要です。

-gl オプションが DBA に設定されている場合は、DBA 権限が必要です。

-gl オプションが NONE に設定されている場合、データのロードは失敗します。

関連する動作

ロードが失敗しても、作成したファイルはそのまま残ります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「sa_describe_shapefile システムプロシージャー」1078 ページ
- 「レッスン 3：ESRI シェイプファイルデータのロード」『SQL Anywhere サーバー 空間データサポート』
- 「LOAD TABLE 文」814 ページ
- 「ESRI シェイプファイルのサポート」『SQL Anywhere サーバー 空間データサポート』

例

次の文は、esri_load テーブルを作成し、架空のシェイプファイルである *c:¥esri¥shapefile.shp* のデータを作成したテーブルにロードし、ロードしたデータを SRID 1000004326 に関連付けます。

```
CALL st_geometry_load_shapefile( 'c:¥¥esri¥¥shapefile.shp', 1000004326, 'esri_load' );
```

xp_cmdshell システムプロシージャー

プロシージャーからオペレーティングシステムコマンドを実行します。

構文

xp_cmdshell(
command
[, *redir_output* | **'no_output'**] **)**

引数

- ***command*** システムコマンドを指定する CHAR(8000) パラメーター。
- ***redir_output*** 出力をコマンドウィンドウに表示するかどうかを指定する任意の CHAR(254) パラメーター。デフォルトの動作では、出力がコマンドウィンドウに表示されます。'no_output' を指定した場合、コマンドウィンドウには出力は表示されません。

備考

xp_cmdshell は、システムコマンドを実行して、制御を元の環境に戻します。xp_cmdshell が返す値は、実行されたシェルプロセスからの終了コードです。子プロセスが起動したときにエラーが発生した場合、戻り値は 2 です。

2 番目のパラメーターは、Windows オペレーティングシステムのコマンドラインアプリケーションにのみ影響します。UNIX の場合は、2 番目のパラメーターの設定にかかわらず、コマンドウィンドウは表示されません。

Windows Mobile の場合は、2 番目のパラメーターの設定にかかわらず、実行したすべてのコマンドがデータベースサーバーメッセージログに表示されます。プロシージャーを実行するには、コンソールシェル *¥¥windows¥cmd.exe* が必要です。

パーミッション

DBA 権限。

参照

- 「CALL 文」503 ページ
- 「セキュリティに関するヒント」『SQL Anywhere サーバー データベース管理』

例

次の文は、現在のディレクトリにあるファイルをファイル *c:¥temp.txt* 内にリストします。

```
CALL xp_cmdshell( 'dir > c:¥¥temp.txt' );
```

次の文は、同じ処理を実行しますが、**[コマンド]** ウィンドウは表示されません。

```
CALL xp_cmdshell( 'dir > c:¥¥temp.txt', 'no_output' );
```

xp_msver システムプロシージャー

データベースサーバーのバージョンと名前についての情報を取り出します。

構文

xp_msver(*the_option* **)**

引数

- ***the_option*** 文字列 (string) は次のいずれかでなければなりません。文字列は文字列デリミターで囲みます。

引数	説明
ProductName	製品 (SQL Anywhere) の名前。
ProductVersion	バージョン番号とそれに続くビルド番号。フォーマットは次のとおりです。 12.0.1.3600
CompanyName	次の文字列を返します。 iAnywhere Solutions, Inc.
FileDescription	製品名、オペレーティングシステム名の順に返します。
LegalCopyright	ソフトウェアの著作権を示す文字列を返します。
LegalTrademarks	ソフトウェアの商標を示す文字列を返します。

備考

xp_msver は、製品、会社、バージョンなどの情報を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

なし。

参照

● 「システム関数」 150 ページ

例

次の文は、バージョンとオペレーティングシステムの説明を要求します。

```
SELECT xp_msver( 'ProductVersion') Version,
  xp_msver( 'FileDescription' ) Description;
```

次に、出力例を示します。Version の値は、システムによって異なります。

Version	Description
12.0.1.3600	SQL Anywhere Windows XP

xp_read_file システムプロシージャー

ファイルを読み込み、そのファイルの内容を LONG BINARY 変数として返します。

構文

xp_read_file(*filename* **[,** *lazy* **])**

引数

- ***filename*** 内容を返す対象ファイルの名前を指定する LONG VARCHAR パラメーター。
- ***lazy*** この任意の INTEGER パラメーターを指定し、その値が 0 以外の場合、ファイルの内容は要求されるまで読み込まれません。読み込みが行われるのは、LONG BINARY 値へのアクセス時のみです。また、その対象はファイル内の要求された部分のみです。デフォルトは 0 (non-lazy) です。

備考

この関数は指定したファイルの内容を読み込んで、結果を LONG BINARY 変数として返します。

filename には、データベースサーバーの開始ディレクトリからの相対ファイル名を指定します。

この関数は、ファイルに保存されているドキュメントやイメージ全体をテーブルに挿入するときに役立ちます。ファイルが読み込めない場合は、関数は NULL を返します。

データファイルの文字セットが異なる場合は、CSCONVERT 関数を使用すると文字セットを変換できます。

また、CSCONVERT 関数を使用すると、xp_read_file システムプロシージャーを使用するときに発生する文字セット変換の要件に対処できます。

パーミッション

DBA 権限。

参照

- 「CSCONVERT 関数 [文字列]」189 ページ
- 「xp_write_file システムプロシージャー」1250 ページ
- 「CALL 文」503 ページ
- 「xp_read_file での openxml の使用」『SQL Anywhere サーバー SQL の使用法』

例

次の文は、テーブル t1 のカラム picture にイメージを挿入します (他のすべてのカラムが NULL を受け入れることができると仮定しています)。

```
INSERT INTO t1 ( picture )
  SELECT xp_read_file( 'portrait.gif' );
```

xp_scanf システムプロシージャー

入力文字列とフォーマット文字列から部分文字列を抽出します。

構文

xp_scanf(
input_buffer,
format,
parm [, *parm2*, ...]
)

引数

- ***input_buffer*** 入力文字列を指定する CHAR(254) パラメーター。
- ***format*** この CHAR(254) パラメーターを使用して、各 *parm* 引数に対する入力文字列のフォーマットをプレースホルダー (%s) を使用して指定します。最高で 50 のプレースホルダーを *format* 引数に指定できます。また、*parm* 引数と同じプレースホルダー数にする必要があります。
- ***parm*** これらの CHAR(254) パラメーターの中から 1 つ以上のパラメーターを使用して、*input_buffer* から抽出する部分文字列を指定します。最大で 50 のパラメーターを指定できます。

備考

xp_scanf システムプロシージャーは、指定された *format* を使用して入力文字列から部分文字列を抽出し、指定された *parm* 値に結果を入力します。

パーミッション

なし。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「CALL 文」503 ページ

例

次の文は、部分文字列の Hello と World! を入力バッファーの Hello World! から抽出します。部分文字列は変数 string1 と string2 に入力された後、選択されます。

```
CREATE VARIABLE string1 CHAR(254);
CREATE VARIABLE string2 CHAR(254);
CALL xp_scanf( 'Hello World!', '%s %s', string1, string2 );
SELECT string1, string2;
```

xp_sendmail システムプロシージャー

電子メールメッセージを送信します。

構文

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*]
)

引数

一部の引数は固定値を指定し、下記のように Transact-SQL との互換性を保つために使用できます。

- **recipient**　受信者のメールアドレスを指定する LONG VARCHAR パラメーター。複数の受信者を指定するときは、各電子メールアドレスをセミコロンで区切ります。

- **subject**　メッセージの件名フィールドを指定する LONG VARCHAR パラメーター。デフォルト値は NULL です。

- **cc_recipient**　cc 受信者のメールアドレスを指定する LONG VARCHAR パラメーター。複数の cc 受信者を指定するときは、各電子メールアドレスをセミコロンで区切ります。デフォルト値は NULL です。

- **bcc_recipient**　bcc 受信者のメールアドレスを指定する LONG VARCHAR パラメーター。複数の bcc 受信者を指定するときは、各電子メールアドレスをセミコロンで区切ります。デフォルト値は NULL です。

- **query**　この LONG VARCHAR は Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルト値は NULL です。

- **"message"**　メッセージの内容を指定する LONG VARCHAR パラメーター。デフォルト値は NULL です。"message" は予約語であるため、"message" パラメーター名は二重引用符で囲む必要があります。

- **attachname**　この LONG VARCHAR パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルト値は NULL です。

- **attach_result**　この INT パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは 0 です。

- **echo_error**　この INT パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは 1 です。

- **include_file**　添付ファイルを指定する LONG VARCHAR パラメーター。デフォルト値は NULL です。

- **no_column_header**　この INT パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは 0 です。

- **'no_output'**　この INT パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは 0 です。

- **width**　この INT パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは 80 です。

- **separator**　この CHAR(1) パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは CHAR(9) です。

- **dbuser**　この LONG VARCHAR パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは guest です。

- **dbname**　この LONG VARCHAR パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは master です。

- **type**　この LONG VARCHAR パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルト値は NULL です。

- **include_query**　この INT パラメーターは Transact-SQL との互換性のために提供されています。SQL Anywhere では使用されません。デフォルトは 0 です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **content_type**　この LONG VARCHAR パラメーターは、"message" パラメーターのコンテンツタイプを指定します (たとえば、text/html、ASIS など)。デフォルト値は NULL です。content_type の値は検証されないため、無効なコンテンツタイプを設定すると、無効な電子メールや理解できない電子メールが送信されます。

  ヘッダーを手動で設定する場合は、content_type パラメーターを ASIS に設定します。この操作を行った場合、xp_sendmail プロシージャーは、"message" パラメーターに渡されたデータが正しく構成されたヘッダー付き電子メールであると想定して、ヘッダーをさらに追加することはしません。ASIS を指定した場合は、データを他のパラメーターに渡すことで通常なら自動的に指定されるヘッダーも含めて、すべてのヘッダーを "message" パラメーターに手動で設定する必要があります。

パーミッション

DBA 権限。

MAPI で電子メールセッションを開始する xp_startmail、または SMTP で電子メールセッションを開始する xp_startsmtp を実行します。

MAPI を使用して電子メールを送信する場合、content_type パラメーターはサポートされません。

備考

xp_sendmail は、xp_startmail または xp_startsmtp でセッションが開始されると、指定された受信者に電子メールメッセージを送信するシステムプロシージャーです。このプロシージャーには、任意の長さのメッセージを指定できます。xp_sendmail の引数値は文字列です。各引数の長さは、システムで使用可能なメモリ容量によって制限されます。

content_type 引数は、MIME 電子メールの要件を理解しているユーザーが使用します。xp_sendmail では、ASIS が content_type として受け入れられます。content_type が ASIS に設定されると、xp_sendmail はメッセージ本文 ("message") が正しく構成されたヘッダー付き電子メールであると想定し、ヘッダーをさらに追加することはしません。ASIS を指定して、複数のコンテンツタイプを含むマルチパートメッセージを送信できます。MIME の詳細については、RFC 2045 ～ 2049 (http://www.ietf.org/) を参照してください。

include_file パラメーターで指定された添付ファイルは application/octet-stream MIME タイプとして base64 エンコードで送信され、データベースサーバーに作成されます。

SQL Anywhere 10.0.0 以降では、SMTP 電子メールシステムで送信される電子メールは、件名行に 7 ビット ASCII ではない文字が含まれていると、エンコードされます。また、SMS 対応デバイスに送信された電子メールは、件名行に 7 ビット ASCII ではない文字が含まれていると、正しくデコードされないことがあります。

リターンコード

「MAPI と SMTP 用のシステムプロシージャーのリターンコード」1031 ページを参照してください。

参照

- 「MAPI プロシージャーと SMTP プロシージャー」1030 ページ
- 「xp_startmail システムプロシージャー」1246 ページ
- 「xp_startsmtp システムプロシージャー」1247 ページ
- 「xp_stopmail システムプロシージャー」1249 ページ
- 「xp_stopsmtp システムプロシージャー」1249 ページ
- 「CALL 文」503 ページ
- 「予約語」1 ページ

例

次の呼び出しは、ファイル *prices.doc* が含まれるメッセージをメール添付書類としてユーザー ID Sales Group に送信します。

```
CALL xp_sendmail( recipient='Sales Group',
     subject='New Pricing',
     include_file = 'C:¥¥DOCS¥¥PRICES.DOC' );
```

次のサンプルプログラムは、内容を見てわかるように、xp_sendmail システムプロシージャーのさまざまな使用例を示します。

```
BEGIN
DECLARE to_list LONG VARCHAR;
DECLARE email_subject CHAR(256);
DECLARE content LONG VARCHAR;
DECLARE uid CHAR(20);

SET to_list='test_account@mytestdomain.com';
SET email_subject='This is a test';
SET uid='test_sender@mytestdomain.com';

// Call xp_startsmtp to start an SMTP email session
CALL xp_startsmtp( uid, 'mymailserver.mytestdomain.com' );

// Basic email example
SET content='This text is the body of my email.¥n';
CALL xp_sendmail( recipient=to_list,
    subject=email_subject,
    "message"=content );

// Send email containing HTML using the content_type parameter,
// as well as including an attachment with the include_file
// parameter
SET content='Plain text.<BR><BR><B>Bold text.</B><BR><BR><a
href="www.iAnywhere.com">iAnywhere
 Home Page</a></B><BR><BR>';
CALL xp_sendmail( recipient=to_list,
    subject=email_subject,
    "message"=content,
    content_type = 'text/html',
    include_file = 'test.zip' );

// Send email "ASIS".  Here the content-type has been specified
// by the user as part of email body.  Note the attachment can
// also be done separately
SET content='Content-Type: text/html;¥nContent-Disposition: inline; ¥n¥nThis text
 is not bold<BR><BR><B>This text is bold</B><BR><BR><a href="www.iAnywhere.com">iAnywhere
Home
 Page</a></B><BR><BR>';
CALL xp_sendmail( recipient=to_list,
```

```
      subject=email_subject,
      "message"=content,
      content_type = 'ASIS',
      include_file = 'test.zip' );

   // Send email "ASIS" along with an include file.  Note that
   // "message" contains the information for another attachment
   SET content = 'Content-Type: multipart/mixed; boundary="xxxxx";¥n';
   SET content = content || 'This part of the email should not be shown.  If this is shown
    then the email client is not MIME compatible¥n¥n';
   SET content = content || '--xxxxx¥n';
   SET content = content || 'Content-Type: text/html;¥n';
   SET content = content || 'Content-Disposition: inline;¥n¥n';
   SET content = content || 'This text is not bold<BR><BR><B>This text is bold</B><BR>
    <BR><a href="www.iAnywhere.com">iAnywhere Home Page</a></B><BR><BR>¥n¥n';
   SET content = content || '--xxxxx¥n';
   SET content = content || 'Content-Type: application/zip; name="test.zip"¥n';
   SET content = content || 'Content-Transfer-Encoding: base64¥n';
   SET content = content || 'Content-Disposition: attachment; filename="test.zip"¥n¥n';

   // Encode the attachment instead of adding this one in
   // the include_file parameter
   SET content = content || base64_encode( xp_read_file( 'othertest.zip' ) ) || '¥n¥n';
   SET content = content || '--xxxxx--¥n';
   CALL xp_sendmail( recipient=to_list,
       subject=email_subject,
       "message"=content,
       content_type = 'ASIS',
       include_file = 'othertest.zip' );

   // End the SMTP session
      CALL xp_stopsmtp();
   END
```

xp_sprintf システムプロシージャー

入力文字列セットから結果文字列を構築します。

構文

xp_sprintf(
output_buffer,
format,
parm [, *parm2*, ...]
)

引数

- ***output_buffer*** 結果文字列を格納する出力バッファーを指定する CHAR(254) パラメーター。
- ***format*** この CHAR(254) パラメーターを使用して、各 *parm* 引数に対する結果文字列のフォーマットをプレースホルダー (%s) を使用して指定します。最高で 50 のプレースホルダーを *format* 引数に指定できます。また、*parm* 引数と同じプレースホルダー数にする必要があります。
- ***parm*** 結果文字列で使用される入力文字列。最高で 50 の CHAR(254) 引数を指定できます。

備考

xp_sprintf システムプロシージャーは、*format* 引数と *parm* 引数を使用して文字列を構築し、結果を *output_buffer* に入力します。

パーミッション

なし。

参照

- 「CALL 文」503 ページ

例

次の文は、文字列 Hello World! を結果の変数に挿入します。

```
CREATE VARIABLE result CHAR(254);
Call xp_sprintf( result, '%s %s', 'Hello', 'World!' );
```

xp_startmail システムプロシージャー

MAPI で電子メールセッションを開始します。

構文

xp_startmail(
[mail_user = *mail-login-name* **]**
[, mail_password = *mail-password* **])**

引数

- ***mail_user*** MAPI ログイン名を指定する LONG VARCHAR パラメーター。
- ***mail_password*** MAPI パスワードを指定する LONG VARCHAR パラメーター。

パーミッション

DBA 権限。

UNIX ではサポートされません。

備考

xp_startmail は、電子メールセッションを開始するシステムプロシージャーです。

Microsoft Exchange を使用する場合、*mail-login-name* 引数は Exchange のプロファイル名を指定します。また、プロシージャーの呼び出しにパスワードを使用しないでください。

リターンコード

「MAPI と SMTP 用のシステムプロシージャーのリターンコード」1031 ページを参照してください。

参照

- 「MAPI プロシージャーと SMTP プロシージャー」1030 ページ
- 「xp_stopmail システムプロシージャー」1249 ページ
- 「xp_sendmail システムプロシージャー」1241 ページ
- 「xp_startsmtp システムプロシージャー」1247 ページ
- 「xp_stopsmtp システムプロシージャー」1249 ページ
- 「CALL 文」503 ページ
- 「セキュリティに関するヒント」『SQL Anywhere サーバー データベース管理』

xp_startsmtp システムプロシージャー

SMTP で電子メールセッションを開始します。

構文

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*]
[**, certificate_company =** *organization*]
[**, certificate_unit =** *organization-unit*]
[**, certificate_name =** *common-name*]
)

引数

- **smtp_sender**　送信者の電子メールアドレスを指定する LONG VARCHAR パラメーター。

- **smtp_server**　使用する SMTP サーバーを指定する LONG VARCHAR パラメーター。これはサーバー名または IP アドレスになります。

- **smtp_port**　SMTP サーバーで接続するポート番号を指定する任意の INTEGER パラメーター。デフォルトは 25 です。

- **timeout**　データサーバーからの応答を待機する秒数を指定する任意の INTEGER パラメーター。この秒数が経過すると、xp_sendmail の現在の呼び出しは中止します。デフォルトは 60 秒です。

- **smtp_sender_name**　送信者の電子メールアドレスのエイリアスを指定する任意の LONG VARCHAR パラメーター。たとえば、'*電子メールアドレス*' の代わりに 'JSmith' などを指定します。

- **smtp_auth_username**　認証を必要とする SMTP サーバーに示すユーザー名を指定する任意の LONG VARCHAR パラメーター。

- **smtp_auth_password**　認証を要求する SMTP サーバーに提供するパスワードを指定する任意の LONG VARCHAR パラメーター。

- **trusted_certificates**　信頼できる証明書を 1 つ以上含むファイルのパスとファイル名を指定する任意の LONG VARCHAR パラメーター。デフォルト値は NULL です。このパラメーターが NULL の場合、標準 SMTP 接続が行われます。

  信頼できる証明書は、サーバーの自己署名証明書、パブリックエンタープライズルート証明書、民間認証局に属する証明書のいずれかです。RSA を使用して証明書を生成してください。

- **certificate_company**　証明書の組織フィールドがこの値と一致する場合にのみクライアントでサーバー証明書が受け入れられるように指定する任意の LONG VARCHAR パラメーター。trusted_certificates 値が NULL の場合、このパラメーターは無視されます。

- **certificate_unit**　証明書の組織単位フィールドがこの値と一致する場合にのみクライアントでサーバー証明書が受け入れられるように指定する任意の LONG VARCHAR パラメーター。trusted_certificates 値が NULL の場合、このパラメーターは無視されます。

- **certificate_name**　証明書の共通名フィールドがこの値と一致する場合にのみクライアントでサーバー証明書が受け入れられるように指定する任意の LONG VARCHAR パラメーター。trusted_certificates 値が NULL の場合、このパラメーターは無視されます。

パーミッション

DBA 権限。

備考

xp_startsmtp は、SMTP サーバーに接続し、指定された電子メールアドレスのメールセッションを開始するシステムプロシージャーです。この接続はタイムアウトになります。そのため、xp_sendmail を実行する直前に xp_startsmtp を呼び出すことをおすすめします。

smtp_auth_username と smtp_auth_password を指定し、サーバーで SMTP 認証機能がサポートされていない場合、エラーコード 104 が返されます。

ウィルススキャナは xp_startsmtp に影響を与え、エラーコード 100 を返す可能性があります。McAfee VirusScan バージョン 8.0.0 以降には、電子メールのワームの大量送信を防止するとともに、xp_sendmail の正しい実行を妨げる設定があります。ウィルススキャンソフトウェアで、大量メール送信防止を回避する処理を指定できる場合は、*dbeng12.exe* と *dbsrv12.exe* を指定します。たとえば、McAfee VirusScan では、この 2 つの処理を **[Properties]** 領域の **[Excluded Processes]** のリストに追加して、大量メール送信を回避できます。

リターンコード

「MAPI と SMTP 用のシステムプロシージャーのリターンコード」1031 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「MAPI プロシージャーと SMTP プロシージャー」1030 ページ
- 「xp_startmail システムプロシージャー」1246 ページ
- 「xp_stopmail システムプロシージャー」1249 ページ
- 「xp_sendmail システムプロシージャー」1241 ページ
- 「xp_stopsmtp システムプロシージャー」1249 ページ
- 「CALL 文」503 ページ
- 「セキュリティに関するヒント」『SQL Anywhere サーバー データベース管理』

xp_stopmail システムプロシージャー

MAPI 電子メールセッションを閉じます。

構文

xp_stopmail( )

パーミッション

DBA 権限。

UNIX ではサポートされません。

備考

xp_stopmail は、電子メールセッションを終了するシステムプロシージャーです。

リターンコード

「MAPI と SMTP 用のシステムプロシージャーのリターンコード」1031 ページを参照してください。

参照

- 「MAPI プロシージャーと SMTP プロシージャー」1030 ページ
- 「xp_startmail システムプロシージャー」1246 ページ
- 「xp_sendmail システムプロシージャー」1241 ページ
- 「xp_startsmtp システムプロシージャー」1247 ページ
- 「xp_stopsmtp システムプロシージャー」1249 ページ
- 「CALL 文」503 ページ
- 「セキュリティに関するヒント」『SQL Anywhere サーバー データベース管理』

xp_stopsmtp システムプロシージャー

SMTP 電子メールセッションを閉じます。

構文

xp_stopsmtp( )

パーミッション

DBA 権限。

備考

xp_stopsmtp は、電子メールセッションを終了するシステムプロシージャーです。

リターンコード

「MAPI と SMTP 用のシステムプロシージャーのリターンコード」1031 ページを参照してください。

参照

- 「MAPI プロシージャーと SMTP プロシージャー」1030 ページ
- 「xp_startmail システムプロシージャー」1246 ページ
- 「xp_stopmail システムプロシージャー」1249 ページ
- 「xp_sendmail システムプロシージャー」1241 ページ
- 「xp_startsmtp システムプロシージャー」1247 ページ
- 「CALL 文」503 ページ
- 「セキュリティに関するヒント」『SQL Anywhere サーバー データベース管理』

xp_write_file システムプロシージャー

SQL 文からデータをファイルに書き込みます。

構文

xp_write_file(
filename,
file_contents
)

引数

- ***filename*** ファイル名を指定する LONG VARCHAR パラメーター。
- ***file_contents*** ファイルに書き込む内容を指定する LONG BINARY パラメーター。

備考

この関数は、*file_contents* をファイル *filename* に書き込みます。成功した場合は 0 を返し、失敗した場合は 0 以外の値を返します。

filename 値には、絶対パスまたは相対パスのプレフィクスを付けることができます。*filename* に相対パスでプレフィクスを付けた場合、ファイル名はデータベースサーバーの作業ディレクトリを基準にします。ファイルがすでに存在する場合は、内容が上書きされます。

この関数は、長いバイナリのデータをファイルにアンロードする場合に便利です。

また、CSCONVERT 関数を使用すると、xp_write_file システムプロシージャーを使用するときに発生する文字セット変換の要件に対処できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パーミッション

DBA 権限。

参照

- 「CSCONVERT 関数 [文字列]」189 ページ
- 「xp_read_file システムプロシージャー」1239 ページ
- 「CALL 文」503 ページ

例

次の例では、xp_write_file を使用して、データ 123456 を含むファイル *accountnum.txt* を作成します。

```
CALL xp_write_file( 'accountnum.txt', '123456' );
```

次の例では、サンプルデータベースの Contacts テーブルに問い合わせて、ニュージャージー在住の各連絡先用にテキストファイルを作成します。各テキストファイルには、連絡先の名前 (GivenName) と姓 (Surname) を連結し、その後に文字列 **.txt** を追加した名前 (たとえば *Reeves_Scott.txt*) が付けられ、連絡先の住所 (Street)、市 (City)、州 (State) が別々の行に格納されます。

```
SELECT xp_write_file(
Surname || '_' ||  GivenName || '.txt',
Street || '¥n' || City || '¥n' || State )
FROM Contacts WHERE State = 'NJ';
```

次の例では、xp_write_file を使用して、Products テーブル内の全製品について画像ファイル (JPG) を作成します。ID カラムの各値は、Photo カラムの対応する値のコンテンツが含まれるファイルのファイル名になります。

```
SELECT xp_write_file( ID || '.jpg' , Photo ) FROM Products;
```

上記の例では、ID は UNIQUE 制約が指定されたローになります。これは、ファイルが以降のローのコンテンツで上書きされないようにするために重要です。また、カラムに格納されているデータに適用できるファイルの拡張子を指定する必要があります。この場合は、Products.Photo に画像データ (JPG) を格納します。

ビュー

システムビュー

カタログには、キーとインデックスの両方からリンクされるシステムテーブルが含まれます。SQL Anywhere の場合、システムテーブルは非表示です。ただし、各テーブルにはシステムビューがあります。一般的に必要とされるジョインを満たすために、システムビューに複数のシステムテーブルのカラムが含まれることもあります。

将来的な SQL Anywhere カタログとの互換性を保つため、アプリケーションではシステムビューを使用し、変化する可能性がある基本となるシステムテーブルは使用しないでください。

ビューと定義の詳細なシステム情報の表示

データベース管理者は、Sybase Central から定義などのシステムビューに関する情報にアクセスできます。

前提条件

DBA 権限。

内容と備考

次のとおりです。

♦ ビューと定義の詳細なシステム情報の表示 (Sybase Central の場合)。

1. SQL Anywhere 12 プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. データベースを右クリックし、**[所有者フィルターの設定]** をクリックします。
3. **[SYS]** をクリックし、**[OK]** をクリックします。
4. 左ウィンドウ枠で、**[ビュー]** をダブルクリックします。
5. 左ウィンドウ枠でビューをクリックし、右ウィンドウ枠で **[SQL]** タブをクリックします。
6. **[データ]** タブをクリックします。

結果

ビュー定義は **[データ]** タブに表示されます。

次の手順

なし。

SYSARTICLE システムビュー

SYSARTICLE システムの各ローは、パブリケーション内のアーティクルを示します。このビューの基本となるシステムテーブルは ISYSARTICLE です。

カラム名	データ型	説明
publication_id	UNSIGNED INT	このアーティクルが含まれるパブリケーション。
table_id	UNSIGNED INT	各アーティクルは、単一のテーブルのカラムとローから構成されます。このカラムには、このテーブルの ID が含まれます。
where_expr	LONG VARCHAR	WHERE 句で定義されたローのサブセットを含むアーティクルの場合、このカラムに探索条件が含まれます。
subscribe_by_expr	LONG VARCHAR	SUBSCRIBE BY 式で定義されたローのサブセットを含むアーティクルの場合、このカラムに式が含まれます。
query	CHAR(1)	データベースサーバーにアーティクルタイプ情報を示します。
alias	VARCHAR(256)	アーティクルのエイリアス。
schema_change_active	BIT	テーブルとパブリケーションが同期スキーマ変更の一部である場合、1 です。

基本となるシステムテーブルに関する制約

PRIMARY KEY (publication_id, table_id)

FOREIGN KEY (publication_id) references SYS.ISYSPUBLICATION (publication_id)

FOREIGN KEY (table_id) references SYS.ISYSTAB (table_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SYSARTICLECOL システムビュー

SYSARTICLECOL システムビューの各ローは、アーティクル内のカラムを識別します。このビューの基本となるシステムテーブルは ISYSARTICLECOL です。

カラム名	データ型	説明
publication_id	UNSIGNED INT	カラムが含まれるパブリケーションのユニークな識別子。
table_id	UNSIGNED INT	カラムが属するテーブル。
column_id	UNSIGNED INT	SYSTABCOL システムビューのカラム識別子。

基本となるシステムテーブルに関する制約

PRIMARY KEY (publication_id, table_id, column_id)

FOREIGN KEY (publication_id, table_id) references SYS.ISYSARTICLE (publication_id, table_id)

FOREIGN KEY (table_id, column_id) references SYS.ISYSTABCOL (table_id, column_id)

SYSCAPABILITY システムビュー

SYSCAPABILITY システムビューの各ローは、リモートデータベースサーバーの機能のステータスを示します。このビューの基本となるシステムテーブルは ISYSCAPABILITY です。

カラム名	データ型	説明
capid	INTEGER	SYSCAPABILITYNAME システムビューに表示されている機能 ID。
srvid	UNSIGNED INT	SYSSERVER システムビューに表示されている、機能が適用されるサーバー。
capvalue	CHAR(128)	機能の値。

基本となるシステムテーブルに関する制約

PRIMARY KEY (capid, srvid)

FOREIGN KEY (srvid) references SYS.ISYSSERVER (srvid)

参照

- 「SYSCAPABILITYNAME システムビュー」1256 ページ

SYSCAPABILITYNAME システムビュー

SYSCAPABILITYNAME システムビューの各ローには、SYSCAPABILITY システムビュー内の各機能 ID の名前があります。

カラム名	データ型	説明
capid	INTEGER	機能をユニークに識別する番号。
capname	VARCHAR(32000)	機能の名前。

備考

SYSCAPABILITYNAME システムビューは、sa_rowgenerator と次のサーバープロパティの組み合わせを使用して定義されます。

- RemoteCapability
- MaxRemoteCapability

参照

- 「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「SYSCAPABILITY システムビュー」1255 ページ

SYSCHECK システムビュー

SYSCHECK システムビューの各ローは、テーブル内の名前付き検査制約を定義します。このビューの基本となるシステムテーブルは ISYSCHECK です。

カラム名	データ型	説明
check_id	UNSIGNED INT	データベースの制約をユニークに識別する番号。
check_defn	LONG VARCHAR	CHECK 式。

基本となるシステムテーブルに関する制約

PRIMARY KEY (check_id)

FOREIGN KEY (check_id) references SYS.ISYSCONSTRAINT (constraint_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SYSCOLPERM システムビュー

GRANT 文を使うと、テーブルの各カラムに UPDATE、SELECT、または REFERENCES パーミッションを与えることができます。UPDATE、SELECT、または REFERENCES パーミッションを持つ各カラムは、SYSCOLPERM システムビューの各ローに記録されます。このビューの基本となるシステムテーブルは ISYSCOLPERM です。

カラム名	データ型	説明
table_id	UNSIGNED INT	カラムが属するテーブルのテーブル番号。
grantee	UNSIGNED INT	カラムにパーミッションを与えられたユーザー ID のユーザー番号。パーミッションの grantee が特別な PUBLIC ユーザー ID のユーザー番号の場合、パーミッションはすべてのユーザー ID に与えられます。
grantor	UNSIGNED INT	パーミッションを付与するユーザー ID のユーザー番号。
column_id	UNSIGNED INT	このカラム番号と table_id を使って、パーミッションを付与されたカラムを識別します。
privilege_type	SMALLINT	このカラムの数値は、カラムパーミッションの種類 (16=REFERENCES、1=SELECT、または 8=UPDATE) を示します。
is_grantable	CHAR(1)	カラムのパーミッションが GRANT OPTION を使って付与されているかどうかを示します。

基本となるシステムテーブルに関する制約

PRIMARY KEY (table_id, grantee, grantor, column_id, privilege_type)

FOREIGN KEY (table_id, column_id) references SYS.ISYSTABCOL (table_id, column_id)

FOREIGN KEY (grantor) references SYS.ISYSUSER (user_id)

FOREIGN KEY (grantee) references SYS.ISYSUSER (user_id)

SYSCOLSTAT システムビュー

SYSCOLSTAT システムビューには、オプティマイザーが使用するヒストグラムなどのカラムの統計情報が含まれます。このビューの内容を取り出すには、sa_get_histogram ストアドプロシージャーまたはヒストグラムユーティリティを使用するのが最適です。このビューの基本となるシステムテーブルは ISYSCOLSTAT です。

カラム名	データ型	説明
table_id	UNSIGNED INT	このカラムが属するテーブルまたはマテリアライズドビューをユニークに識別する番号。
column_id	UNSIGNED INT	table_id とともに使用してカラムをユニークに識別する番号。
format_id	SMALLINT	システムでのみ使用。
update_time	TIMESTAMP	このカラムの統計情報が最後に更新された時刻。
density	FLOAT	カラムの単一の値の平均による選択性の推定。ローに格納されている大きい単一値の選択性は考慮されません。
max_steps	SMALLINT	システムでのみ使用。
actual_steps	SMALLINT	システムでのみ使用。
step_values	LONG BINARY	システムでのみ使用。
frequencies	LONG BINARY	システムでのみ使用。

注意
SQL Anywhere 12 以降を使用して作成されたデータベースでは、不正なアクセスからデータを保護するため、このビューの基本となるシステムテーブルは常に暗号化されます。

基本となるシステムテーブルに関する制約

PRIMARY KEY (table_id, column_id)

FOREIGN KEY (table_id, column_id) references SYS.ISYSTABCOL (table_id, column_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SYSCONSTRAINT システムビュー

SYSCONSTRAINT システムビューの各ローは、データベース内の名前付き制約を示します。このビューの基本となるシステムテーブルは ISYSCONSTRAINT です。

カラム名	データ型	説明
constraint_id	UNSIGNED INT	制約のユニークな ID。
constraint_type	CHAR(1)	制約の型。 ● **C** カラム検査制約 ● **T** テーブル制約 ● **P** プライマリキー ● **F** 外部キー ● **U** 一意性制約
ref_object_id	UNSIGNED BIGINT	制約の適用対象となるカラム、テーブル、インデックスのオブジェクト ID。
table_object_id	UNSIGNED BIGINT	スクリプトの適用対象となるテーブルのオブジェクト ID。
constraint_name	CHAR(128)	制約名。

基本となるシステムテーブルに関する制約

PRIMARY KEY (constraint_id)

FOREIGN KEY (ref_object_id) references SYS.ISYSOBJECT (object_id)

FOREIGN KEY (table_object_id) references SYS.ISYSOBJECT (object_id)

UNIQUE Constraint (table_object_id, constraint_name)

SYSDBFILE システムビュー

SYSDBFILE システムビューの各ローは、DB 領域ファイルを示します。このビューの基本となるシステムテーブルは ISYSDBFILE です。

カラム名	データ型	説明
dbfile_id	SMALLINT	内部でのみ使用。

カラム名	データ型	説明
dbspace_id	SMALLINT	データベースの各 DB 領域ファイルには、ユニークな番号が割り当てられています。システム DB 領域には、すべてのシステムオブジェクトが含まれ、0 の dbspace_id があります。
dbfile_name	CHAR(128)	DB 領域のファイル名。システムと TEMPORARY 以外の DB 領域では、ファイル名は次の文を使用して変更できます。 ALTER DBSPACE dbspace RENAME 'new-filename';
file_name	LONG VARCHAR	DB 領域のユニークなファイル名。これは CREATE TABLE コマンド中で使われます。
lob_map	LONG VARBIT	内部でのみ使用。

基本となるシステムテーブルに関する制約

PRIMARY KEY (dbfile_id)

FOREIGN KEY (dbspace_id) references SYS.ISYSDBSPACE (dbspace_id)

UNIQUE Index (file_name)

SYSDBSPACE システムビュー

SYSDBSPACE システムビューの各ローは、DB 領域ファイルを示します。このビューの基本となるシステムテーブルは ISYSDBSPACE です。

カラム名	データ型	説明
dbspace_id	SMALLINT	DB 領域を識別するユニークな番号。システム DB 領域には、すべてのシステムオブジェクトが含まれ、0 の dbspace_id があります。
object_id	UNSIGNED BIGINT	DB 領域のオブジェクト ID。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
dbspace_name	CHAR(128)	DB 領域のユニークなファイル名。これは CREATE TABLE コマンド中で使われます。
store_type	TINYINT	内部でのみ使用。

基本となるシステムテーブルに関する制約

PRIMARY KEY (dbspace_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

SYSDBSPACEPERM システムビュー

SYSDBSPACEPERM システムビューの各ローは、DB 領域ファイルのパーミッションを示します。このビューの基本となるシステムテーブルは ISYSDBSPACEPERM です。

カラム名	データ型	説明
dbspace_id	SMALLINT	DB 領域を識別するユニークな番号。システム DB 領域には、すべてのシステムオブジェクトが含まれ、0 の dbspace_id があります。
grantee	UNSIGNED INT	パーミッションを取得するユーザーのユーザー ID。
privilege_type	SMALLINT	grantee に付与されるパーミッション。たとえば、CREATE は DB 領域でオブジェクトを作成するパーミッションを grantee に付与します。

基本となるシステムテーブルに関する制約

FOREIGN KEY (dbspace_id) references SYS.ISYSDBSPACE (dbspace_id)

FOREIGN KEY (grantee) references SYS.ISYSUSER (user_id)

参照

- 「GRANT 文」782 ページ
- 「ユーザー ID、権限、パーミッション」『SQL Anywhere サーバー データベース管理』

SYSDEPENDENCY システムビュー

SYSDEPENDENCY システムビューの各ローは、2 つのデータベースオブジェクト間の依存性を示します。このビューの基本となるシステムテーブルは ISYSDEPENDENCY です。

あるオブジェクトが定義内の別のオブジェクトを参照する場合、この 2 つのデータベースオプション間に依存性が存在します。たとえば、ビューのクエリ指定がテーブルを参照する場合、ビューはテーブルに依存していると呼びます。データベースサーバーが、テーブル、ビュー、マテリアライズドビュー、カラムの依存性を追跡します。

カラム名	データ型	説明
ref_object_id	UNSIGNED BIGINT	参照オブジェクトのオブジェクト ID。
dep_object_id	UNSIGNED BIGINT	参照オブジェクトの ID。

基本となるシステムテーブルに関する制約

PRIMARY KEY (ref_object_id, dep_object_id)

FOREIGN KEY (ref_object_id) references SYS.ISYSOBJECT (object_id)

FOREIGN KEY (dep_object_id) references SYS.ISYSOBJECT (object_id)

参照

- 「sa_dependent_views システムプロシージャー」 1070 ページ
- 「ビューの依存性」『SQL Anywhere サーバー SQL の使用法』

SYSDOMAIN システムビュー

SYSDOMAIN システムビューは、組み込みデータ型に関する情報 (ドメインとも呼びます) を記録します。通常の操作ではビューのコンテンツは変化しません。このビューの基本となるシステムテーブルは ISYSDOMAIN です。

カラム名	データ型	説明
domain_id	SMALLINT	各データ型に割り当てられたユニークな番号。この番号は変更できません。
domain_name	CHAR(128)	CREATE TABLE コマンド中に見られるデータ型の名前 (CHAR または INTEGER など)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
type_id	SMALLINT	ODBC データ型。この値は、Transact-SQL 互換の dbo.SYSTYPES テーブルにある data_type の値に対応します。
"precision"	SMALLINT	このデータ型を使って格納できる有効桁数。数値でないデータ型に対応するカラム値は NULL です。

基本となるシステムテーブルに関する制約

PRIMARY KEY (domain_id)

SYSEVENT システムビュー

SYSEVENT システムビューの各ローは、CREATE EVENT で作成されたイベントを示します。このビューの基本となるシステムテーブルは ISYSEVENT です。

カラム名	データ型	説明
event_id	UNSIGNED INT	各イベントに割り当てられたユニークな番号。
object_id	UNSIGNED BIGINT	データベースのイベントをユニークに識別するイベントの内部 ID。
creator	UNSIGNED INT	イベントの所有者のユーザー番号。ユーザー名は SYSUSER システムビューで確認できます。
event_name	VARCHAR(128)	イベント名。
enabled	CHAR(1)	イベントが起動できるかどうかを示します。
location	CHAR(1)	イベントを起動するロケーション。 ● C = 統合 ● R = リモート ● A = すべて

カラム名	データ型	説明
event_type_id	UNSIGNED INT	システムイベントの場合、イベント型は SYSEVENTTYPE システムビューにリストされます。
action	LONG VARCHAR	イベントハンドラー定義。難読化された値は、隠されたイベントを示します。
external_action	LONG VARCHAR	システムでのみ使用。
condition	LONG VARCHAR	イベントハンドラーの起動を制御するのに使用される条件。
remarks	LONG VARCHAR	イベントの注釈。このカラムは ISYSREMARK に由来します。
source	LONG VARCHAR	イベントの元のソース。このカラムは ISYSSOURCE に由来します。

基本となるシステムテーブルに関する制約

PRIMARY KEY (event_id)

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

UNIQUE Index (event_name)

参照

- 「SYSEVENTTYPE システムビュー」1264 ページ

SYSEVENTTYPE システムビュー

SYSEVENTTYPE システムビューは、CREATE EVENT で参照できるシステムのイベントタイプを定義します。

カラム名	データ型	説明
event_type_id	INT	各イベント型に割り当てられたユニークな番号。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
name	VARCHAR(32000)	システムイベント型の名前。
description	LONG VARCHAR	システムイベント型の説明。

備考

SYSEVENTTYPE システムビューは、sa_rowgenerator と次のサーバープロパティの組み合わせを使用して定義されます。

- EventTypeName
- EventTypeDesc
- MaxEventType

参照

- 「データベースサーバープロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「SYSEVENT システムビュー」1263 ページ

SYSEXTERNENV システムビュー

C/C++ で記述された Embedded SQL と ODBC アプリケーション、Java、Perl、PHP、または Microsoft .NET Framework Common Language Runtime (CLR) に基づく C# や Visual Basic などの言語で記述されたアプリケーションを含む、多くの外部ランタイム環境がサポートされています。

SYSEXTERNENV システムビューの各ローは、各外部環境を識別して起動するために必要な情報を示します。このビューの基本となるシステムテーブルは ISYSEXTERNENV です。

カラム名	データ型	説明
object_id	unsigned bigint	外部環境のユニークな識別子。
name	char(128)	外部環境または言語の名前。
scope	char(1)	外部環境が接続ごとに 1 つ (C) 起動されるか、データベースごとに 1 つ (D) 起動されるかを識別します。
support_result_sets	char(1)	ユーザーに結果セットを返す可能性のある外部環境を識別します。

カラム名	データ型	説明
location	long varchar	環境のメイン実行プログラムを検出できるロケーションを識別します。
options	long varchar	外部環境を起動するためにコマンドラインで必要なオプションを識別します。
user_id	unsigned int	内部でのみ使用。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (user_id) references SYS.ISYSUSER (user_id)

UNIQUE Index (name)

SYSEXTERNENVOBJECT システムビュー

C/C++ で記述された Embedded SQL と ODBC アプリケーション、Java、Perl、PHP、または Microsoft .NET Framework Common Language Runtime (CLR) に基づく C# や Visual Basic などの言語で記述されたアプリケーションを含む、多くの外部ランタイム環境がサポートされています。

SYSEXTERNENVOBJECT システムビューの各ローは、インストールされた外部オブジェクトを示します。このビューの基本となるシステムテーブルは ISYSEXTERNENVOBJECT です。

カラム名	データ型	説明
object_id	unsigned bigint	外部オブジェクトのユニークな識別子。
extenv_id	unsigned bigint	外部環境のユニークな識別子 (SYSEXTERNENV.object_id)。
owner	unsigned int	外部オブジェクトの作成者／所有者を識別します。
name	long varchar	INSTALL 文に指定された外部オブジェクトの名前を識別します。
contents	long binary	外部オブジェクトの内容。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
update_time	timestamp	オブジェクトが最後に変更 (またはインストール) された時刻を識別します。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (extenv_id) references SYS.ISYSEXTERNENV (object_id)

FOREIGN KEY (owner) references SYS.ISYSUSER (user_id)

UNIQUE Index (name)

SYSEXTERNLOGIN システムビュー

SYSEXTERNLOGIN システムビューの各ローは、リモートデータアクセスの外部ログインを示します。このビューの基本となるシステムテーブルは ISYSEXTERNLOGIN です。

注意
SYSEXTERNLOGINS システムテーブルに含まれる前のカタログバージョン。このテーブル名は ISYSEXTERNLOGIN ('S' なし) に変更され、このビューの基本となるテーブルになります。

カラム名	データ型	説明
user_id	UNSIGNED INT	ローカルデータベースでのユーザー ID。
srvid	UNSIGNED INT	SYSSERVER システムビューにリストされているリモートサーバー。
remote_login	VARCHAR(128)	このユーザーのリモートサーバーのログイン名。
remote_password	VARBINARY(128)	このユーザーのリモートサーバーのパスワード。

注意
SQL Anywhere 12 を使用して作成されたデータベースでは、不正なアクセスからデータを保護するため、このビューの基本となるシステムテーブルは常に暗号化されます。

基本となるシステムテーブルに関する制約

PRIMARY KEY (user_id, srvid)

FOREIGN KEY (user_id) references SYS.ISYSUSER (user_id)

FOREIGN KEY (srvid) references SYS.ISYSSERVER (srvid)

SYSFKEY システムビュー

SYSFKEY システムビューの各ローは、システム内の外部キー制約を示します。このビューの基本となるシステムテーブルは ISYSFKEY です。

カラム名	データ型	説明
foreign_table_id	UNSIGNED INT	外部テーブルのテーブル番号。
foreign_index_id	UNSIGNED INT	外部キーのインデックス番号。
primary_table_id	UNSIGNED INT	プライマリテーブルのテーブル番号。
primary_index_id	UNSIGNED INT	プライマリキーのインデックス番号。
match_type	TINYINT	制約と一致する型。一致する型には次の型があります。 ● **0** デフォルトのマッチングを使用 ● **1** SIMPLE ● **2** FULL ● **129** SIMPLE UNIQUE ● **130** FULL UNIQUE 一致する型の詳細については、「CREATE TABLE 文」651 ページの MATCH 句を参照してください。
check_on_commit	CHAR(1)	COMMIT で外部キーが有効であるかどうかが確認されるまで、INSERT と UPDATE 文を待たせるかどうかを示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
nulls	CHAR(1)	外部キーのカラムが NULL 値を許容するかどうかを示します。この設定は外部キーのカラム中の nulls 設定とは独立していることに注意してください。

基本となるシステムテーブルに関する制約

PRIMARY KEY (foreign_table_id, foreign_index_id)

FOREIGN KEY (foreign_table_id, foreign_index_id) references SYS.ISYSIDX (table_id, index_id)

FOREIGN KEY (primary_table_id, primary_index_id) references SYS.ISYSIDX (table_id, index_id)

SYSGROUP システムビュー

SYSGROUP システムビューには、各グループの各メンバーに 1 つのローがあります。このビューは、グループとメンバーの多対多の関係を示します。グループには複数のメンバーがあり、あるユーザーは複数のグループのメンバーになれます。このビューの基本となるシステムテーブルは ISYSGROUP です。

カラム名	データ型	説明
group_id	UNSIGNED INT	グループのユーザー番号。
group_member	UNSIGNED INT	メンバーのユーザー番号。

基本となるシステムテーブルに関する制約

PRIMARY KEY (group_id, group_member)

FOREIGN KEY group_id (group_id) references SYS.ISYSUSER (user_id)

FOREIGN KEY group_member (group_member) references SYS.ISYSUSER (user_id)

SYSHISTORY システムビュー

SYSHISTORY システムビューの各ローは、データベースの開始、データベースの調整など、データベースに対するシステム操作を記録します。このビューの基本となるシステムテーブルは ISYSHISTORY です。

カラム名	データ型	説明
operation	CHAR(128)	データベースファイルに対して実行されるオペレーションのタイプ。operation には次のいずれかの値を指定します。 ● **INIT** データベースがいつ作成されたかに関する情報。 ● **UPGRADE** データベースがいつアップグレードされたかに関する情報。 ● **START** 特定のオペレーティングシステム上で特定のバージョンのデータベースサーバーを使用してデータベースがいつ開始されたかに関する情報。 ● **LAST_START** データベースサーバーが最後に開始された時刻に関する情報。LAST_START オペレーションは、LAST_START ローに現在格納されている値とは異なるバージョンのデータベースサーバーか異なるオペレーティングシステム、またはその両方でデータベースが開始されたときに START オペレーションに変換されます。 ● **DTT** DB 領域で実行される **2 番目から最後**のディスク転送時間 (DTT: Disk Transfer Time) の調整操作に関する情報。つまり、ALTER DATABASE CALIBRATE 文か ALTER DATABASE RESTORE DEFAULT CALIBRATION 文のどちらかの最後から

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
		2 番目の実行に関する情報です。 ● **LAST_DTT** DB 領域で実行される**最新の** DTT 調整操作に関する情報。つまり、ALTER DATABASE CALIBRATE 文か ALTER DATABASE RESTORE DEFAULT CALIBRATION 文のどちらかの最新の実行に関する情報です。 ● **LAST_BACKUP** 最後のバックアップ (バックアップの日時を含みます)、バックアップの種類、バックアップするファイル、バックアップを実行したデータベースサーバーのバージョンに関する情報。
object_id	UNSIGNED INT	DTT と LAST_DTT 以外の操作については、このカラムの値は 0 になります。DTT と LAST_DTT の操作の場合、SYSDBSPACE システムビューで定義されている DB 領域の dbspace_id です。

カラム名	データ型	説明
sub_operation	CHAR(128)	DTT と LAST_DTT 以外の操作については、このカラムの値は、空の単一引用符 (') のセットになります。DTT と LAST_DTT の操作の場合、このカラムには、DB 領域で実行されるサブ操作の種類も含まれます。次のような値があります。 ● **DTT_SET** DB 領域の調整が設定されます。 ● **DTT_UNSET** DB 領域のデフォルト設定が復元されます。
version	CHAR(128)	オペレーションの実行に使用されるデータベースサーバーのバージョンとビルド番号。
platform	CHAR(128)	オペレーションが実行されたオペレーティングシステム。
first_time	TIMESTAMP	特定のオペレーティングシステム上の特定のバージョンのソフトウェアでデータベースが最初に開始された日時。
last_time	TIMESTAMP	特定のオペレーティングシステム上の特定のバージョンのソフトウェアでデータベースが最後に開始された日時。
details	LONG VARCHAR	このカラムは、データベースサーバーの起動に使用されたコマンドラインオプションやデータベースに対して有効になっている機能ビットなどの情報を格納します。この情報は、Sybase 製品の保守契約を結んでいるサポートセンタが使用します。

基本となるシステムテーブルに関する制約

PRIMARY KEY (operation, object_id, version, platform)

参照

- 「SYSDBSPACE システムビュー」1260 ページ

SYSIDX システムビュー

SYSIDX システムビューの各ローは、データベースの論理インデックスを定義します。このビューの基本となるシステムテーブルは ISYSIDX です。

カラム名	データ型	説明
table_id	UNSIGNED INT	このインデックスが適用されるテーブルをユニークに識別します。
index_id	UNSIGNED INT	テーブル内のインデックスを識別するユニークな番号。
object_id	UNSIGNED BIGINT	データベースのインデックスをユニークに識別するインデックスの内部 ID。
phys_index_id	UNSIGNED INT	論理インデックスの実装に使用する基本となる物理インデックスを識別します。この値は、テンポラリテーブルまたはリモートテーブル上のインデックスに対しては NULL となります。その他の場合、この値は SYSPHYSIDX システムビューの物理インデックスの object_id に対応します。
dbspace_id	SMALLINT	インデックスを含むファイルの ID。この値は SYSDBSPACE システムビューのエントリに対応します。

カラム名	データ型	説明
file_id	SMALLINT	廃止予定。このカラムは SYSVIEW に存在しますが、基本となるシステムテーブル ISYSIDX には存在しません。このカラムの内容は dbspace_id と同じであり、互換性のために提供されています。今後は dbspace_id を使用してください。
index_category	TINYINT	インデックスの型。次のような値があります。 ● **1** プライマリキー ● **2** 外部キー ● **3** セカンダリインデックス (一意性制約を含む) ● **4** テキストインデックス
"unique"	TINYINT	インデックスがユニークインデックス (1) か、ユニークでないインデックス (4) か、一意性制約 (2) かを示します。ユニークインデックスでは、インデックスカラムの 2 つのローは同じ値を持てません。
index_name	CHAR(128)	インデックスの名前。
not_enforced	CHAR(1)	システムでのみ使用。
file_id	SMALLINT	システムでのみ使用。

基本となるシステムテーブルに関する制約

PRIMARY KEY (table_id, index_id)

FOREIGN KEY (table_id) references SYS.ISYSTAB (table_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (table_id, phys_index_id) references SYS.ISYSPHYSIDX (table_id, phys_index_id)

UNIQUE Index (index_name, table_id, index_category)

参照

- 「SYSIDXCOL システムビュー」1275 ページ
- 「SYSPHYSIDX システムビュー」1285 ページ
- 「SYSDBSPACE システムビュー」1260 ページ

SYSIDXCOL システムビュー

SYSIDXCOL システムビューの各ローは、SYSIDX システムビューで記述されているインデックスのカラム 1 つを示します。このビューの基本となるシステムテーブルは ISYSIDXCOL です。

カラム名	データ型	説明
table_id	UNSIGNED INT	インデックスが適用されるテーブルを識別します。
index_id	UNSIGNED INT	カラムが適用されるインデックスを識別します。table_id と index_id は共同で SYSIDX システムビューにある 1 つのインデックスを識別します。
sequence	SMALLINT	インデックス内の各カラムには、0 から始まるユニークな番号が割り当てられます。この番号はインデックス内のカラムの相対的な重要度を示します。もっとも重要なカラムの sequence 番号は 0 になります。
column_id	UNSIGNED INT	インデックスを作成するテーブルのカラムを識別します。table_id と column_id は共同で、SYSCOLUMN システムビューで表される 1 つのカラムを識別します。
"order"	CHAR(1)	インデックス内のカラムが昇順 (A) か、降順 (D) かを示します。テキストインデックスの場合、この値は NULL です。

カラム名	データ型	説明
primary_column_id	UNSIGNED INT	この外部キーカラムに対応するプライマリキーの ID。非外部キーのカラムの場合、値は NULL です。

基本となるシステムテーブルに関する制約

PRIMARY KEY (table_id, index_id, column_id)

FOREIGN KEY (table_id, index_id) references SYS.ISYSIDX (table_id, index_id)

FOREIGN KEY (table_id, column_id) references SYS.ISYSTABCOL (table_id, column_id)

参照

- 「SYSIDX システムビュー」1273 ページ

SYSJAR システムビュー

SYSJAR システムビューの各ローは、データベースに格納されている JAR ファイルを定義します。このビューの基本となるシステムテーブルは ISYSJAR です。

カラム名	データ型	説明
jar_id	INTEGER	JAR ファイルを識別するユニークな番号。
object_id	UNSIGNED BIGINT	データベースのインデックスをユニークに識別する JAR ファイルの内部 ID。
creator	UNSIGNED INT	JAR ファイルの作成者のユーザー番号。
jar_name	LONG VARCHAR	JAR ファイルの名前。
jar_file	LONG VARCHAR	データベース内の JAR ファイルの外部ファイル名。
update_time	TIMESTAMP	JAR ファイルが最後に更新された時刻。

基本となるシステムテーブルに関する制約

PRIMARY KEY (jar_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

UNIQUE Index (jar_name)

参照

- 「SYSJARCOMPONENT システムビュー」1277 ページ

SYSJARCOMPONENT システムビュー

SYSJAR システムビューの各ローは、JAR ファイルコンポーネントを定義します。このビューの基本となるシステムテーブルは ISYSJARCOMPONENT です。

カラム名	データ型	説明
component_id	INTEGER	コンポーネントの ID を含むプライマリキー。
jar_id	INTEGER	JAR の ID 番号を含むフィールド。
component_name	LONG VARCHAR	コンポーネント名。
component_type	CHAR(1)	コンポーネントの型。
contents	LONG BINARY	JAR ファイルのバイトコード。

基本となるシステムテーブルに関する制約

PRIMARY KEY (component_id)

FOREIGN KEY (jar_id) references SYS.ISYSJAR (jar_id)

参照

- 「SYSJAR システムビュー」1276 ページ

SYSJAVACLASS システムビュー

SYSJAVACLASS システムビューの各ローは、データベースに格納されている Java クラス 1 つを示します。このビューの基本となるシステムテーブルは ISYSJAVACLASS です。

カラム名	データ型	説明
class_id	INTEGER	Java クラスのユニークな番号。テーブルのプライマリキーでもあります。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	データベースのインデックスをユニークに識別する Java クラスの内部 ID。
creator	UNSIGNED INT	クラスの作成者のユーザー番号。
jar_id	INTEGER	クラスがある JAR ファイルの ID。
class_name	LONG VARCHAR	Java クラスの名前。
public	CHAR(1)	クラスがパブリック (Y) かプライベート (N) かを示します。
component_id	INTEGER	SYSJARCOMPONENT システムビューのコンポーネントの ID。
update_time	TIMESTAMP	クラスが最後に更新された時刻。

基本となるシステムテーブルに関する制約

PRIMARY KEY (class_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

FOREIGN KEY (component_id) references SYS.ISYSJARCOMPONENT (component_id)

SYSLOGINMAP システムビュー

SYSLOGINMAP システムビューには、統合化ログインまたは Kerberos ログインを使用してデータベースに接続できる各ユーザーに 1 つのローが含まれます。セキュリティ上の理由から、このビューの内容は DBA 権限を持つユーザーだけが表示できます。このビューの基本となるシステムテーブルは ISYSLOGINMAP です。

カラム名	データ型	説明
login_mode	TINYINT	ログインの種類：統合化ログインの場合は 1、Kerberos ログインの場合は 2。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
login_id	VARCHAR(1024)	database_uid にマッピングする統合化ログインのユーザープロファイル名または Kerberos 原則。
object_id	UNSIGNED BIGINT	ユニークな識別子。ユーザー ID とデータベースのユーザー ID 間のマッピングごとに 1 つ。
database_uid	UNSIGNED INT	ログイン ID をマッピングするデータベースユーザー ID。

基本となるシステムテーブルに関する制約

PRIMARY KEY (login_mode, login_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (database_uid) references SYS.ISYSUSER (user_id)

SYSLOGINPOLICY システムビュー

このビューの基本となるシステムテーブルは ISYSLOGINPOLICY です。

カラム名	データ型	説明
login_policy_id	UNSIGNED BIGINT	ログインポリシーのユニークな識別子。
login_policy_name	CHAR(128)	ログインポリシーの名前。

基本となるシステムテーブルに関する制約

PRIMARY KEY (login_policy_id)

FOREIGN KEY (login_policy_id) references SYS.ISYSOBJECT (object_id)

UNIQUE Index (login_policy_name)

参照

- 「SYSLOGINPOLICYOPTION システムビュー」1280 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSLOGINPOLICYOPTION システムビュー

このビューの基本となるシステムテーブルは ISYSLOGINPOLICYOPTION です。

カラム名	データ型	説明
login_policy_id	UNSIGNED BIGINT	ログインポリシーのユニークな識別子。
login_option_name	CHAR(128)	ログインポリシーの名前。
login_option_value	LONG VARCHAR	ログインポリシーの作成時の値。

基本となるシステムテーブルに関する制約

PRIMARY KEY (login_policy_id, login_option_name)

FOREIGN KEY (login_policy_id) references SYS.ISYSLOGINPOLICY (login_policy_id)

参照

- 「SYSLOGINPOLICY システムビュー」1279 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSMIRROROPTION システムビュー

このビューの基礎となるシステムテーブルは ISYSMIRROROPTION です。

カラム名	データ型	説明
option_name	CHAR(128)	オプションの名前。
option_value	LONG VARCHAR	ミラーが作成された時点のオプション値。

基本となるシステムテーブルに関する制約

PRIMARY KEY (option_name)

参照

- 「SYSMIRRORSERVER システムビュー」1280 ページ
- 「SYSMIRRORSERVEROPTION システムビュー」1281 ページ

SYSMIRRORSERVER システムビュー

このビューの基礎となるシステムテーブルは ISYSMIRRORSERVER です。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	ミラーサーバーのユニークな識別子。
server_name	CHAR(128)	サーバーの名前。
server_type	CHAR(20)	サーバーのタイプ。値は、PRIMARY、MIRROR、ARBITER、PARTNER、または COPY のいずれかです。
parent	UNSIGNED BIGINT	親サーバー。値が NULL の場合、サーバーはデータベースミラーリングシステムのプライマリサーバーまたはミラーサーバーです。 このカラムに値がある場合は、現在のサーバーの親サーバーの ID です。
alternate_parent	UNSIGNED BIGINT	現在の親が使用できなくなった場合に、代替の親として使用されるサーバーの ID です。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id)

UNIQUE Index (server_name)

参照

- 「SYSMIRROROPTION システムビュー」1280 ページ
- 「SYSMIRRORSERVEROPTION システムビュー」1281 ページ
- 「コピーノードの親の判断」『SQL Anywhere サーバー データベース管理』
- 「CREATE MIRROR SERVER 文」579 ページ

SYSMIRRORSERVEROPTION システムビュー

このビューの基礎となるシステムテーブルは ISYSMIRRORSERVEROPTION です。

カラム名	データ型	説明
server_id	UNSIGNED BIGINT	ミラーサーバーのユニークな識別子。
option_name	CHAR(128)	オプションの名前。
option_value	LONG VARCHAR	ミラーが作成された時点のオプション値。

基本となるシステムテーブルに関する制約

PRIMARY KEY (server_id, option_name)

FOREIGN KEY (server_id) references SYS.ISYSMIRRORSERVER (object_id)

参照

- 「SYSMIRROROPTION システムビュー」1280 ページ
- 「SYSMIRRORSERVER システムビュー」1280 ページ

SYSMVOPTION システムビュー

SYSMVOPTION システムビューの各ローは、マテリアライズドビューまたはテキストインデックスを作成した時点のオプション値に関する記述です。オプションの名前は、SYSMVOPTIONNAME システムビューにあります。このビューの基本となるシステムテーブルは ISYSMVOPTION です。

カラム名	データ型	説明
view_object_id	UNSIGNED BIGINT	マテリアライズドビューのオブジェクト ID。
option_id	UNSIGNED INT	データベースのオプションを識別するユニークな番号。オプション名を参照するには、SYSMVOPTIONNAME システムビューを参照してください。
option_value	LONG VARCHAR	マテリアライズドビューが作成された時点のオプション値。

基本となるシステムテーブルに関する制約

PRIMARY KEY (view_object_id, option_id)

FOREIGN KEY (view_object_id) references SYS.ISYSOBJECT (object_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

FOREIGN KEY (option_id) references SYS.ISYSMVOPTIONNAME (option_id)

参照

- 「SYSMVOPTIONNAME システムビュー」1283 ページ
- 「データベース内のテキストインデックスに関する情報の表示 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』
- 「高度：カタログ内のマテリアライズドビュー情報の表示」『SQL Anywhere サーバー SQL の使用法』

SYSMVOPTIONNAME システムビュー

SYSMVOPTION システムビューの各ローには、マテリアライズドビューまたはテキストインデックスを作成した時点の名前オプション値があります。オプションの値は、SYSMVOPTION システムビューにあります。このビューの基本となるシステムテーブルは ISYSMVOPTIONNAME です。

カラム名	データ型	説明
option_id	UNSIGNED INT	データベースのオプションをユニークに識別する番号。
option_name	CHAR(128)	オプションの名前。

基本となるシステムテーブルに関する制約

PRIMARY KEY (option_id)

UNIQUE Index (option_name)

参照

- 「SYSMVOPTION システムビュー」1282 ページ
- 「データベース内のテキストインデックスに関する情報の表示 (Sybase Central の場合)」『SQL Anywhere サーバー SQL の使用法』
- 「高度：カタログ内のマテリアライズドビュー情報の表示」『SQL Anywhere サーバー SQL の使用法』

SYSOBJECT システムビュー

SYSOBJECT システムビューの各ローは、データベースオブジェクトを示します。このビューの基本となるシステムテーブルは ISYSOBJECT です。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	データベースのオブジェクトをユニークに識別するインデックスの内部 ID。
status	TINYINT	オブジェクトのステータス。次のような値があります。 ● **1 (有効)**　オブジェクトは、データベースサーバーから使用できます。このステータスは、ENABLED と同等です。つまり、オブジェクトを ENABLE にすると、ステータスは VALID に変更されます。 ● **2 (無効)**　内部操作後に、オブジェクトを再コンパイルする試みが失敗しました。たとえば、依存するオブジェクトを変更するスキーマの変更後などです。データベースサーバーは、文で参照されるオブジェクトを再コンパイルする試みを継続します。 ● **4 (無効化)**　ユーザーがオブジェクトを、明示的に無効化しました。たとえば、ALTER TABLE...DISABLE VIEW DEPENDENCIES 文を使用する場合などです。
object_type	TINYINT	オブジェクトの種類。
creation_time	TIMESTAMP	オブジェクトを作成した日付と時刻。
object_type_str	CHAR(128)	オブジェクトの種類。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SYSOPTION システムビュー

SYSOPTION システムビューには、データベースに格納されている各オプション設定のローのオプションが含まれます。各ユーザーはオプションごとに自分の設定を保存できます。また、ユーザー ID PUBLIC に対する設定は、自分の設定を持たないユーザーが使うデフォルトの設定になります。このビューの基本となるシステムテーブルは ISYSOPTION です。

カラム名	データ型	説明
user_id	UNSIGNED INT	このオプション設定が適用されるユーザー番号。
"option"	CHAR(128)	オプションの名前。
"setting"	LONG VARCHAR	現在のオプションの設定。

基本となるシステムテーブルに関する制約

PRIMARY KEY (user_id, "option")

FOREIGN KEY (user_id) references SYS.ISYSUSER (user_id)

SYSOPTSTAT システムビュー

SYSOPTSTAT システムビューは、コストモデルの調整情報を、ALTER DATABASE CALIBRATE 文で計算して格納します。このビューのコンテンツは内部使用のみです。sa_get_dtt システムプロシージャー経由でアクセスすることをおすすめします。このビューの基本となるシステムテーブルは ISYSOPTSTAT です。

カラム名	データ型	説明
stat_id	UNSIGNED INT	システムでのみ使用。
group_id	UNSIGNED INT	システムでのみ使用。
format_id	SMALLINT	システムでのみ使用。
data	LONG BINARY	システムでのみ使用。

基本となるシステムテーブルに関する制約

PRIMARY KEY (stat_id, group_id, format_id)

SYSPHYSIDX システムビュー

SYSPHYSIDX システムビューの各ローは、データベースの物理インデックスを定義します。このビューの基本となるシステムテーブルは ISYSPHYSIDX です。

カラム名	データ型	説明
table_id	UNSIGNED INT	インデックスが対応するテーブルのオブジェクト ID。
phys_index_id	UNSIGNED INT	テーブル内の物理インデックスのユニークな番号。
root	INTEGER	データベースファイルにおける物理インデックスのルートページの位置を識別します。
key_value_count	UNSIGNED INT	インデックス内の個別のキー値の数。
leaf_page_count	UNSIGNED INT	リーフインデックスページの数。
depth	UNSIGNED SMALLINT	物理インデックスの深さ (レベル数)。
max_key_distance	UNSIGNED INT	システムでのみ使用。
seq_transitions	UNSIGNED INT	システムでのみ使用。
rand_transitions	UNSIGNED INT	システムでのみ使用。
rand_distance	UNSIGNED INT	システムでのみ使用。
allocation_bitmap	LONG VARBIT	システムでのみ使用。
long_value_bitmap	LONG VARBIT	システムでのみ使用。

基本となるシステムテーブルに関する制約

PRIMARY KEY (table_id, phys_index_id)

参照

- 「SYSIDX システムビュー」 1273 ページ
- 「SYSIDXCOL システムビュー」 1275 ページ

SYSPROCEDURE システムビュー

SYSPROCEDURE システムビューの各ローは、データベース内のプロシージャーを示します。このビューの基本となるシステムテーブルは ISYSPROCEDURE です。

カラム名	データ型	説明
proc_id	UNSIGNED INT	各プロシージャーにはユニークな番号 (プロシージャー番号) が割り当てられます。
creator	UNSIGNED INT	プロシージャーの所有者。
object_id	UNSIGNED BIGINT	データベースのプロシージャーをユニークに識別するプロシージャーの内部 ID。
proc_name	CHAR(128)	プロシージャー名。1 人の作成者が、同じ名前のプロシージャーを 2 つ持つことはできません。
proc_defn	LONG VARCHAR	プロシージャーの定義。
remarks	LONG VARCHAR	プロシージャーに関する注記。ISYSREMARK システムテーブル内に格納されている値。
replicate	CHAR(1)	このプロパティは内部でのみ使用されます。
srvid	UNSIGNED INT	プロシージャーがリモートデータベースサーバー上のプロシージャーのプロキシである場合は、そのリモートサーバーを表します。
source	LONG VARCHAR	回避されたプロシージャーのソース。この値は、ISYSSOURCE システムテーブル内に格納されています。
avg_num_rows	FLOAT	プロシージャーが FROM 句に表示されるときに、クエリの最適化に使用するために収集される情報。
avg_cost	FLOAT	プロシージャーが FROM 句に表示されるときに、クエリの最適化に使用するために収集される情報。

カラム名	データ型	説明
stats	LONG BINARY	プロシージャーが FROM 句に表示されるときに、クエリの最適化に使用するために収集される情報。

基本となるシステムテーブルに関する制約

PRIMARY KEY (proc_id)

FOREIGN KEY (srvid) references SYS.ISYSSERVER (srvid)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

UNIQUE Index (proc_name, creator)

SYSPROCPARM システムビュー

SYSPROCPARM システムビューの各ローは、データベース内のプロシージャーに対するパラメーター 1 つを示します。このビューの基本となるシステムテーブルは ISYSPROCPARM です。

カラム名	データ型	説明
proc_id	UNSIGNED INT	このパラメーターが属するプロシージャーをユニークに識別します。
parm_id	SMALLINT	各プロシージャーは、パラメーターに 1 から順に番号を付けます。パラメーター番号の順は、定義された順になっています。関数の場合、最初のパラメーターの内容は関数名であり、その関数の戻り値を表します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
parm_type	SMALLINT	パラメーターは、次に示すタイプのいずれかに該当します。 ● **0** 標準のパラメーター (変数) ● **1** 結果変数 ー 結果セットを返すプロシージャーで使用 ● **2** SQLSTATE エラー値 ● **3** SQLCODE エラー値 ● **4** 関数からの戻り値
parm_mode_in	CHAR(1)	このパラメーターが、プロシージャーに値を提供するかどうかを示します (IN または INOUT パラメーター)。
parm_mode_out	CHAR(1)	このパラメーターが、プロシージャーからの値 (IN または INOUT パラメーター) と RESULT 句のカラムのどちらを返すかを示します。
domain_id	SMALLINT	SYSDOMAIN システムビューにリストされたデータ型番号から、パラメーターのデータ型を識別します。
width	BIGINT	文字列パラメーターでは長さ、数値パラメーターでは精度、その他のデータ型では記憶領域のサイズをバイトで示します。
scale	SMALLINT	数値データ型の場合、小数点以下の桁数です。その他のデータ型の場合、このカラムの値は 1 です。
user_type	SMALLINT	パラメーターのユーザー型 (適用できる場合)。

カラム名	データ型	説明
parm_name	CHAR(128)	プロシージャーパラメーターの名前。
"default"	LONG VARCHAR	パラメーターのデフォルト値。参照情報としてのみ表示されます。
remarks	LONG VARCHAR	常に NULL を返します。ODBC ドライバーの旧バージョンを新しいパーソナルデータベースサーバーで使用できるようにするために用意されています。
base_type_str	VARCHAR(32767)	パラメーターの物理的な型を表す注釈付きの型文字列。

基本となるシステムテーブルに関する制約

PRIMARY KEY (proc_id, parm_id)

FOREIGN KEY (proc_id) references SYS.ISYSPROCEDURE (proc_id)

FOREIGN KEY (domain_id) references SYS.ISYSDOMAIN (domain_id)

FOREIGN KEY (user_type) references SYS.ISYSUSERTYPE (type_id)

SYSPROCPERM システムビュー

SYSPROCPERM システムビューの各ローは、あるプロシージャーを実行するための、ユーザーに付与されたパーミッションを示します。パーミッションを付与されたユーザーだけがプロシージャーを実行できます。このビューの基本となるシステムテーブルは ISYSPROCPERM です。

カラム名	データ型	説明
proc_id	UNSIGNED INT	プロシージャー番号は、パーミッションが付与されたプロシージャーをユニークに特定します。
grantee	UNSIGNED INT	パーミッションを受けるユーザーのユーザー番号。

基本となるシステムテーブルに関する制約

PRIMARY KEY (proc_id, grantee)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

FOREIGN KEY (grantee) references SYS.ISYSUSER (user_id)

FOREIGN KEY (proc_id) references SYS.ISYSPROCEDURE (proc_id)

SYSPROXYTAB システムビュー

SYSPROXYTAB システムビューの各ローは、プロキシテーブルのリモートパラメーターを示します。このビューの基本となるシステムテーブルは ISYSPROXYTAB です。

カラム名	データ型	説明
table_object_id	UNSIGNED BIGINT	プロキシテーブルのオブジェクト ID。
existing_obj	CHAR(1)	プロキシテーブルがリモートサーバーに以前に存在していたかどうかを示します。
srvid	UNSIGNED INT	プロキシテーブルと関連付けられたリモートサーバーのユニークな ID。
remote_location	LONG VARCHAR	リモートサーバーに関するプロキシテーブルの位置。

基本となるシステムテーブルに関する制約

PRIMARY KEY (table_object_id)

FOREIGN KEY (table_object_id) references ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (srvid) references SYS.ISYSSERVER (srvid)

SYSPUBLICATION システムビュー

SYSPUBLICATION システムビューの各ローは、SQL Remote または Mobile Link パブリケーションを示します。このビューの基本となるシステムテーブルは ISYSPUBLICATION です。

カラム名	データ型	説明
publication_id	UNSIGNED INT	パブリケーションをユニークに識別する番号。
object_id	UNSIGNED BIGINT	データベースのパブリケーションをユニークに識別するプロシージャーの内部 ID。

カラム名	データ型	説明
creator	UNSIGNED INT	パブリケーションの所有者。
publication_name	CHAR(128)	パブリケーションの名前。
remarks	LONG VARCHAR	パブリケーションに関する注記。ISYSREMARK システムテーブル内に格納されている値。
type	CHAR(1)	このカラムは使用されません。
sync_type	UNSIGNED INT	パブリケーションの同期の種類。次のような値があります。 ● **0 (ログスキャン)** トランザクションログを使用して、最後のアップロード後に変更されたすべての関連データをアップロードする通常のパブリケーションです。 ● **1 (スクリプト化されたアップロード)** このパブリケーションの場合、トランザクションログは無視され、アップロードは格納済みプロシージャーを使用してユーザーが定義します。ストアドプロシージャーに関する情報は、ISYSSYNCSCRIPT システムテーブルに格納されます。 ● **2 (ダウンロード専用)** これはダウンロードのみのパブリケーションです。データはアップロードされません。

基本となるシステムテーブルに関する制約

PRIMARY KEY (publication_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

UNIQUE Index (publication_name, creator)

参照

- 「スクリプト化されたアップロード」『Mobile Link クライアント管理』
- 「SYSSYNCSCRIPT システムビュー」1310 ページ

SYSREMARK システムビュー

SYSREMARK システムビューの各ローは、オブジェクトの注釈 (コメント) を示します。このビューの基本となるシステムテーブルは ISYSREMARK です。

カラム	データ型	説明
object_id	UNSIGNED BIGINT	関連する注釈があるオブジェクトの内部 ID。
remarks	LONG VARCHAR	オブジェクトと関連付けられた注釈またはコメント。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

SYSREMOTEOPTION システムビュー

SYSREMOTEOPTION システムビューの各ローは、SQL Remote メッセージリンクパラメーターの値を示します。このビューの基本となるシステムテーブルは ISYSREMOTEOPTION です。

このビューの一部のカラムには、機密データが含まれている可能性があります。このため、このビューにアクセスできるのは DBA 権限を持つユーザーに限られます。SYSREMOTEOPTION2 ビューを使用すると、機密データを含む可能性のあるカラムを除いて、このビューのデータにパブリックアクセスできます。

カラム	データ型	説明
option_id	UNSIGNED INT	メッセージリンクパラメーターの ID 番号。
user_id	UNSIGNED INT	パラメーターが設定されているユーザー ID。

カラム	データ型	説明
"setting"	VARCHAR(255)	メッセージリンクパラメーターの値。

基本となるシステムテーブルに関する制約

PRIMARY KEY (option_id, user_id)

FOREIGN KEY (option_id) references SYS.ISYSREMOTEOPTIONTYPE (option_id)

FOREIGN KEY (user_id) references SYS.ISYSUSER (user_id)

SYSREMOTEOPTIONTYPE システムビュー

SYSREMOTEOPTIONTYPE システムビューの各ローは、SQL Remote メッセージリンクパラメーター 1 つを示します。このビューの基本となるシステムテーブルは ISYSREMOTEOPTIONTYPE です。

カラム	データ型	説明
option_id	UNSIGNED INT	メッセージリンクパラメーターの ID 番号。
type_id	SMALLINT	このパラメーターを使用するメッセージタイプの ID 番号。
"option"	VARCHAR(128)	メッセージリンクパラメーターの名前。

基本となるシステムテーブルに関する制約

PRIMARY KEY (option_id)

FOREIGN KEY (type_id) references SYS.ISYSREMOTETYPE (type_id)

SYSREMOTETYPE システムビュー

SYSREMOTETYPE システムビューには、SQL Remote に関する情報があります。このビューの基本となるシステムテーブルは ISYSREMOTETYPE です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
type_id	SMALLINT	SQL Remote によってサポートされているメッセージシステムのうち、このユーザーにメッセージを送信するために使用されているものを識別します。
object_id	UNSIGNED BIGINT	データベースのリモートタイプをユニークに識別するインデックスの内部 ID。
type_name	CHAR(128)	SQL Remote がサポートするメッセージシステムの名前。
publisher_address	LONG VARCHAR	リモートデータベースパブリッシャーのアドレス。
remarks	LONG VARCHAR	リモートタイプに関する注釈。ISYSREMARK システムテーブル内に格納されている値。

基本となるシステムテーブルに関する制約

PRIMARY KEY (type_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

UNIQUE Index (type_name)

SYSREMOTEUSER システムビュー

SYSREMOTEUSER システムビューの各ローは、REMOTE パーミッションを持つユーザー ID (サブスクライバー) を示します。そのユーザーに送信された SQL Remote メッセージと、そのユーザーから送信されたメッセージのステータスも合わせて示します。このビューの基本となるシステムテーブルは ISYSREMOTEUSER です。

カラム名	データ型	説明
user_id	UNSIGNED INT	REMOTE パーミッションを持つユーザーのユーザー番号。

カラム名	データ型	説明
consolidate	CHAR(1)	ユーザーに CONSOLIDATE パーミッション (Y) か REMOTE パーミッション (N) が付与されていることを表します。
type_id	SMALLINT	SQL Remote によってサポートされているメッセージシステムのうち、このユーザーにメッセージを送信するために使用されているものを識別します。
address	LONG VARCHAR	SQL Remote メッセージが送信されるアドレス。アドレスは address_type に対して適切でなければいけません。
frequency	CHAR(1)	SQL Remote メッセージが送信される頻度。
send_time	TIME	次にメッセージがこのユーザーへ送信される時刻。
log_send	UNSIGNED BIGINT	メッセージは、log_send が log_sent よりも大きいサブスクライバーだけに送信されます。
time_sent	TIMESTAMP	このサブスクライバーに、最後にメッセージが送信された時刻。
log_sent	UNSIGNED BIGINT	最後に送信されたオペレーションのログオフセット。
confirm_sent	UNSIGNED BIGINT	このサブスクライバーから最後に確認されたオペレーションに対する、ログオフセット。
send_count	INTEGER	SQL Remote メッセージが送信された回数。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
resend_count	INTEGER	メッセージがサブスクライバーデータベースに、一度だけ適用されたことを確認するためのカウンター。
time_received	TIMESTAMP	このサブスクライバーから、最後にメッセージが受信された時刻。
log_received	UNSIGNED BIGINT	現在のデータベースで最後に受信されたオペレーションに対する、サブスクライバーのデータベース内のログオフセット。
confirm_received	UNSIGNED BIGINT	最後に確認メッセージが送信されたオペレーションに対する、サブスクライバーのデータベース内のログオフセット。
receive_count	INTEGER	メッセージが受信された回数。
rereceive_count	INTEGER	メッセージが現在のデータベースに、一度だけ適用されたことを確認するためのカウンター。

基本となるシステムテーブルに関する制約

PRIMARY KEY (user_id)

FOREIGN KEY (user_id) references SYS.ISYSUSER (user_id)

FOREIGN KEY (type_id) references SYS.ISYSREMOTETYPE (type_id)

UNIQUE Index (type_id, address)

SYSSCHEDULE システムビュー

SYSSCHEDULE システムビューの各ローは、CREATE EVENT 文の SCHEDULE 句に指定されたイベントの起動時刻を示します。このビューの基本となるシステムテーブルは ISYSSCHEDULE です。

カラム名	データ型	説明
event_id	UNSIGNED INT	各イベントに割り当てられたユニークな番号。
sched_name	VARCHAR(128)	イベントのスケジュールと関連付けられた名前。
recurring	TINYINT	スケジュールを反復するかどうかを示します。
start_time	TIME	スケジュールの起動時刻。
stop_time	TIME	BETWEEN を指定した場合、スケジュールの停止時刻を示します。
start_date	DATE	イベントの実行がスケジュールされている最初の日付。
days_of_week	TINYINT	イベントがスケジュールされている曜日を示すビットマスク。 ● x01 = 日曜日 ● x02 = 月曜日 ● x04 = 火曜日 ● x08 = 水曜日 ● x10 = 木曜日 ● x20 = 金曜日 ● x40 = 土曜日
days_of_month	UNSIGNED INT	イベントがスケジュールされている日付を示すビットマスク。いくつかの例を示します。 ● x01 = 1 日 ● x02 = 2 日 ● x40000000 = 31 日 ● x80000000 = その月の最終日

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
interval_units	CHAR(10)	EVERY で指定される間隔単位。 ● HH = 時間 ● NN = 分 ● SS = 秒
interval_amt	INTEGER	EVERY で指定される期間。

基本となるシステムテーブルに関する制約

PRIMARY KEY (event_id, sched_name)

FOREIGN KEY (event_id) references SYS.ISYSEVENT (event_id)

SYSSEQUENCE システムビュー

SYSSEQUENCE システムビューには、各ユーザー定義シーケンスごとに 1 つのローがあります。このビューの基礎となるシステムテーブルは ISYSSEQUENCE です。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	各シーケンスに割り当てられたユニークな番号。
owner	UNSIGNED INT	シーケンスの所有者。
min_value	BIGINT	シーケンスに許可されている最小値です。
max_value	BIGINT	シーケンスに許可されている最大値です。
increment_by	BIGINT	シーケンスの増分値。
start_with	BIGINT	シーケンスの開始値。
cache	UNSIGNED INT	より速いアクセスのためにメモリに事前割り付けされるシーケンス値の数です。値が 0 の場合は、値が事前割り付けされないことを示します。

カラム名	データ型	説明
cycle	TINYINT	最大値または最小値に達した後に、値の生成を継続するかどうか。
resume_at	BIGINT	ALTER SEQUENCE 文によって指定された RESTART WITH 値です。ALTER RESTART WITH 文が実行されていない場合、値は NULL です。
sequence_name	CHAR(128)	シーケンスの名前。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT MATCH UNIQUE FULL

FOREIGN KEY (owner) references SYS.ISYSUSER (user_id)

SYSSEQUENCEPERM システムビュー

SYSSEQUENCEPERM システムビューには、ユーザーまたはグループがシーケンスに対して保持する権限が記録されます。このビューの基礎となるシステムテーブルは ISYSSEQUENCEPERM です。

カラム名	データ型	説明
sequence_id	UNSIGNED BIGINT	各シーケンスに割り当てられたユニークな番号。
grantee	UNSIGNED INT	シーケンスの変更または削除の権限を持つユーザーまたはグループの ID です。
grantor	UNSIGNED INT	シーケンスに対する権限を付与されたユーザーの ID です。
privilege_type	SMALLINT	ユーザーまたはグループに付与されたシーケンスの権限のタイプです。

基本となるシステムテーブルに関する制約

PRIMARY KEY (sequence_id, grantee, privilege_type)

FOREIGN KEY (sequence_id) references SYS.ISYSSEQUENCE (object_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

FOREIGN KEY (grantee) references SYS.ISYSUSER (user_id)

FOREIGN KEY (grantor) references SYS.ISYSUSER (user_id)

SYSSERVER システムビュー

SYSSERVER システムビューの各ローは、リモートサーバーを示します。このビューの基本となるシステムテーブルは ISYSSERVER です。

注意
SYSSERVERS システムテーブルに含まれる前のカタログバージョン。このテーブル名は ISYSSERVER ('S' なし) に変更され、このビューの基本となるテーブルになります。

カラム名	データ型	説明
srvid	UNSIGNED INT	リモートサーバーの識別子。
srvname	VARCHAR(128)	リモートサーバーの名前。
srvclass	LONG VARCHAR	CREATE SERVER 文で指定されたサーバークラス。
srvinfo	LONG VARCHAR	サーバー情報。
srvreadonly	CHAR(1)	サーバーが読み込み専用かどうか。

基本となるシステムテーブルに関する制約

PRIMARY KEY (srvid)

SYSSOURCE システムビュー

SYSSOURCE システムビューの各ローには、SYSOBJECT システムビューにリストされているオブジェクトのソースコード (適用できる場合) が含まれます。このビューの基本となるシステムテーブルは ISYSSOURCE です。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	ソースコードが定義されているオブジェクトの内部 ID。

カラム名	データ型	説明
source	LONG VARCHAR	オブジェクトを作成したときに preserve_source_format データベースオプションが On の場合、このカラムにはオブジェクトの元のソースコードが含まれます。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

参照

- 「preserve_source_format オプション」『SQL Anywhere サーバー データベース管理』

SYSSPATIALREFERENCESYSTEM システムビュー

SYSSPATIALREFERENCESYSTEM システムビューの各ローは、データベースに定義されている SRS に関する記述です。このビューの基礎となるシステムテーブルは ISYSSPATIALREFERENCESYSTEM です。

このビューには、ST_SPATIAL_REFERENCE_SYSTEMS システムビューとは若干異なる量の情報があります。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	システムでのみ使用。
owner	UNSIGNED INT	SRS の所有者。
srs_name	CHAR(128)	SRS の名前
srs_id	INT	空間参照系の数値識別子 (SRID)。
round_earth	CHAR(1)	SRS タイプが ROUND EARTH (Y) または PLANAR (N) のいずれであるか。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
axis_order	CHAR(12)	データベースサーバーが緯度と経度に関連してポイントを解釈する方法を記述します (たとえば、ST_Lat メソッドと ST_Long メソッドを使用する場合)。非地理的空間参照系の場合、軸順序は x/y/z/m です。地理的空間参照系の場合、デフォルトの軸順序は long/lat/z/m であり、lat/long/z/m もサポートされます。
snap_to_grid	DOUBLE	SQL Anywhere が計算を実行するときに使用するグリッドのサイズを定義します。
tolerance	DOUBLE	ポイントを比較するときに使用する精度を定義します。
semi_major_axis	DOUBLE	ROUND EARTH SRS の楕円の中心から赤道までの距離です。
semi_minor_axis	DOUBLE	ROUND EARTH SRS の楕円の中心から両極までの距離です。
inv_flattening	DOUBLE	ROUND EARTH SRS の楕円に使用される逆扁平率です。 逆扁平率 (f) は、回転楕円体の極から赤道への扁平度を定義する算術値です。値の範囲は、扁平なし (完全な円) から完全な扁平 (直線) までです。逆扁平率は、次に示すように、値 1/f です。 1/f = (semi_major_axis) / (semi_major_axis - semi_minor_axis)
min_x	DOUBLE	座標において許される x の最小値です。
max_x	DOUBLE	座標において許される x の最大値です。

カラム名	データ型	説明
min_y	DOUBLE	座標において許される y の最小値です。
max_y	DOUBLE	座標において許される y の最大値です。
min_z	DOUBLE	座標において許される z の最小値です。
max_z	DOUBLE	座標において許される z の最大値です。
min_m	DOUBLE	座標において許される m の最小値です。
max_m	DOUBLE	座標において許される m の最大値です。
organization	LONG VARCHAR	空間参照系によって使用される座標系を作成した組織の名前。
organization_coordsys_id	INT	座標系を作成した組織が座標系に対して指定した ID。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
srs_type	CHAR(11)	SQL/MM 標準による定義に従った SRS のタイプ。次のいずれかの値を取ります。 ● **GEOGRAPHIC** 緯度と経度 (および標高) の軸を持つ、測地参照される座標系に基づく SRS 用です。これらの SRS は、PLANAR タイプまたは ROUND EARTH タイプです。 ● **PROJECTED** 緯度と経度の軸を持たず、測地参照される座標系に基づく SRS 用です。これらの SRS は PLANAR タイプです。 ● **ENGINEERING** 測地参照されない座標系に基づく SRS 用です。これらの SRS は PLANAR タイプです。 ● **GEOCENTRIC** サポートされていません。 ● **COMPOUND** サポートされていません。 ● **VERTICAL** サポートされていません。 srs_type が空の場合、タイプは指定されません。
linear_unit_of_measure	UNSIGNED BIGINT	空間参照系によって使用される線形測定単位です。
angular_unit_of_measure	UNSIGNED BIGINT	空間参照系によって使用される角度測定単位です。
count_in_use	UNSIGNED BIGINT	内部でのみ使用。

カラム名	データ型	説明
polygon_format	LONG VARCHAR	多角形のリングの方向。CounterClockwise、ClockWise、または EvenOdd のいずれかです。
storage_format	LONG VARCHAR	データが正規化フォーマット (Internal)、非正規化フォーマット (Original)、または両方 (Mixed) のいずれで格納されるか。
definition	LONG VARCHAR	OGC 標準によって定義されているフォーマットでの、空間参照系の WKT 定義です。
transform_definition	LONG VARCHAR	この SRS から別の SRS にデータを変換するときに使用される変換定義設定です。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id)

FOREIGN KEY (linear_unit_of_measure) references SYS.ISYSUNITOFMEASURE (object_id)

FOREIGN KEY (angular_unit_of_measure) references SYS.ISYSUNITOFMEASURE (object_id)

FOREIGN KEY (owner) references SYS.ISYSUSER (user_id)

UNIQUE Constraint (srs_name)

UNIQUE Constraint (srs_id)

参照

- 「ST_SPATIAL_REFERENCE_SYSTEMS 統合ビュー」1336 ページ
- 「CREATE SPATIAL REFERENCE SYSTEM 文」635 ページ

SYSSQLSERVERTYPE システムビュー

SYSSQLSERVERTYPE システムビューには、Adaptive Server Enterprise との互換性に関する情報が含まれます。このビューの基本となるシステムテーブルは ISYSSQLSERVERTYPE です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
ss_user_type	SMALLINT	Adaptive Server Enterprise ユーザー型。
ss_domain_id	SMALLINT	Adaptive Server Enterprise のドメイン ID。
ss_type_name	VARCHAR(30)	Adaptive Server Enterprise 型名。
primary_sa_domain_id	SMALLINT	対応する SQL Anywhere のプライマリドメイン ID。
primary_sa_user_type	SMALLINT	対応する SQL Anywhere のプライマリユーザータイプ。

基本となるシステムテーブルに関する制約

PRIMARY KEY (ss_user_type)

SYSSUBSCRIPTION システムビュー

SYSSUBSCRIPTION システムビューの各ローは、REMOTE パーミッションを持つあるユーザー ID からの、あるパブリケーションに対するサブスクリプションを示します。このビューの基本となるシステムテーブルは ISYSSUBSCRIPTION です。

カラム名	データ型	説明
publication_id	UNSIGNED INT	ユーザー ID のサブスクリプションが作成されているパブリケーションの識別子。
user_id	UNSIGNED INT	パブリケーションに対して送信されたユーザーの ID。
subscribe_by	CHAR(128)	サブスクリプション用の SUBSCRIBE BY 式がある場合、その値。
created	UNSIGNED BIGINT	トランザクションログ内の、サブスクリプションが作成されたオフセット。
started	UNSIGNED BIGINT	トランザクションログ内の、サブスクリプションが開始されたオフセット。

基本となるシステムテーブルに関する制約

PRIMARY KEY (publication_id, user_id, subscribe_by)

FOREIGN KEY (publication_id) references SYS.ISYSPUBLICATION (publication_id)

FOREIGN KEY (user_id) references SYS.ISYSUSER (user_id)

SYSSYNC システムビュー

SYSSYNC システムビューには、Mobile Link 同期に関する情報が含まれます。このビューの一部のカラムには、機密データが含まれている可能性があります。このため、このビューにアクセスできるのは DBA 権限を持つユーザーに限られます。SYSSYNC2 ビューを使用すると、機密データを含む可能性のあるカラムを除いて、このビューのデータにパブリックアクセスできます。このビューの基本となるシステムテーブルは ISYSSYNC です。

カラム名	データ型	説明
sync_id	UNSIGNED INT	ユニークにローを識別する番号。
type	CHAR(1)	値は常に D です。
publication_id	UNSIGNED INT	SYSPUBLICATION システムビューに入っている publication_id。
progress	UNSIGNED BIGINT	最後に成功したアップロードのログオフセット。
site_name	CHAR(128)	Mobile Link ユーザー名。
"option"	LONG VARCHAR	同期オプション。
server_connect	LONG VARCHAR	Mobile Link サーバーのアドレスまたは URL。
server_conn_type	LONG VARCHAR	同期時に使用する、TCP/IP などの通信プロトコル。
last_download_time	TIMESTAMP	最後に Mobile Link サーバーからダウンロードストリームを受信した時刻。
last_upload_time	TIMESTAMP	最後に情報のアップロードが成功した時刻 (Mobile Link サーバーで測定)。デフォルトは jan-1-1900 です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
created	UNSIGNED BIGINT	サブスクリプションが作成されたログオフセット。
log_sent	UNSIGNED BIGINT	情報をどこまでアップロードしたかを示すログの進行状況。更新されるこのカラムのエントリに対するアップロード確認の受信は必要ありません。
generation_number	INTEGER	ファイルベースのダウンロードで、このサブスクリプションに対して最後に受信した世代番号。デフォルトは 0。
extended_state	VARCHAR(1024)	内部でのみ使用。
script_version	CHAR(128)	CREATE SYNCHRONIZATION SUBSCRIPTION 文、ALTER SYNCHRONIZATION SUBSCRIPTION 文、および START SYNCHRONIZATION SCHEMA CHANGE 文によって使用されるスクリプトバージョンを示します。
subscription_name	CHAR(128)	サブスクリプションの名前。

基本となるシステムテーブルに関する制約

PRIMARY KEY (sync_id)

FOREIGN KEY (publication_id) references SYS.ISYSPUBLICATION (publication_id)

UNIQUE Index (publication_id, site_name)

UNIQUE Index (subscription_name)

SYSSYNCPROFILE システムビュー

SYSSYNCPROFILE システムビューには、Mobile Link 同期の同期プロファイルに関する情報が含まれます。

このビューの基本となるシステムテーブルは ISYSSYNCPROFILE です。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	同期プロファイルのオブジェクト ID。
profile_name	CHAR(128)	同期プロファイルの名前。
profile_defn	LONG VARCHAR	同期プロファイルの定義。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id)

UNIQUE Index (profile_name)

SYSSYNCSCRIPT システムビュー

SYSSYNCSCRIPT システムビューの各ローは、Mobile Link のスクリプト化されたアップロードに関するストアドプロシージャーを識別します。このビューは、SYSSYNCSCRIPTS ビューとほとんど同じですが、このビューの値は未加工形式である点が異なります。

このビューの基本となるシステムテーブルは ISYSSYNCSCRIPT です。

カラム名	データ型	説明
pub_object_id	UNSIGNED BIGINT	スクリプトが所属するパブリケーションのオブジェクト ID。
table_object_id	UNSIGNED BIGINT	スクリプトの適用対象となるテーブルのオブジェクト ID。
type	UNSIGNED INT	アップロードプロシージャーのタイプ。
proc_object_id	UNSIGNED BIGINT	パブリケーションに使用するストアドプロシージャーのオブジェクト ID。

基本となるシステムテーブルに関する制約

PRIMARY KEY (pub_object_id, table_object_id, type)

FOREIGN KEY (pub_object_id) references SYS.ISYSOBJECT (object_id)

FOREIGN KEY (table_object_id) references SYS.ISYSOBJECT (object_id)

FOREIGN KEY (proc_object_id) references SYS.ISYSOBJECT (object_id)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「スクリプト化されたアップロード」『Mobile Link クライアント管理』
- 「SYSSYNCSCRIPTS 統合ビュー」1352 ページ
- 「SYSPROCEDURE システムビュー」1286 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ

SYSTAB システムビュー

SYSTAB システムビューの各ローは、データベース内のテーブル 1 つを示します。ビューの追加情報が SYSVIEW システムビューにあります。このビューの基本となるシステムテーブルは ISYSTAB です。

カラム名	データ型	説明
table_id	UNSIGNED INT	各テーブルにはユニークな番号 (テーブル番号) が割り当てられます。
dbspace_id	SMALLINT	テーブルを含む DB 領域を示す値。
count	UNSIGNED BIGINT	テーブルまたはマテリアライズドビュー内のロー数。この値は、各チェックポイントが正常処理されているときに更新されます。この数を使用してデータベースアクセスが最適化されます。非マテリアライズドビューの場合、またはリモートテーブルの場合、カウントは常に 0 です。
creator	UNSIGNED INT	このユーザー番号はテーブルまたはビューの所有者を示します。
table_page_count	INTEGER	基本となるテーブルで使用されるメインページの総数。
ext_page_count	INTEGER	このテーブルで使用される拡張ページの総数。

カラム名	データ型	説明
commit_action	INTEGER	グローバルテンポラリテーブルの場合、0 は、テーブルの作成時に ON COMMIT PRESERVE ROWS 句が指定されたことを示します。1 は、テーブルの作成時に ON COMMIT DELETE ROWS 句が指定されたことを示します (テンポラリテーブルのデフォルト動作)。3 は、テーブルの作成時に NOT TRANSACTIONAL 句が指定されたことを示します。非テンポラリテーブルの場合、commit_action は常に 0 です。
share_type	INTEGER	グローバルテンポラリテーブルの場合、4 は、テーブルの作成時に SHARE BY ALL 句が指定されたことを示します。5 は、テーブルの作成時に SHARE BY ALL 句が指定され**なかった**ことを示します。非テンポラリテーブルの場合、share_type は常に 5 です。これは、非テンポラリテーブルの作成時に SHARE BY ALL 句は指定できないためです。
object_id	UNSIGNED BIGINT	テーブルのオブジェクト ID。
last_modified_at	TIMESTAMP	テーブルが最後に修正された日時。このカラムは、チェックポイント時間にのみ更新されます。
last_modified_tsn	UNSIGNED BIGINT	テーブルを修正したトランザクションに割り当てられたシーケンス番号。このカラムは、チェックポイント時にのみ更新されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
file_id	SMALLINT	廃止予定。このカラムは SYSVIEW に存在しますが、基本となるシステムテーブル ISYSTAB には存在しません。このカラムの内容は dbspace_id と同じであり、互換性のために提供されています。今後は dbspace_id を使用してください。
table_name	CHAR(128)	テーブルまたはビューの名前。1 人の作成者が、同じ名前のテーブルまたはビューを 2 つ以上持つことはできません。
table_type	TINYINT	テーブルまたはビューのタイプ。次のような値があります。 ● **1** ベーステーブル ● **2** マテリアライズドビュー ● **3** グローバルテンポラリテーブル ● **4** ローカルテンポラリテーブル ● **5** テキストインデックスベーステーブル ● **6** テキストインデックスグローバルテンポラリテーブル ● **21** ビュー
replicate	CHAR(1)	この値は内部でのみ使用されます。

カラム名	データ型	説明
server_type	TINYINT	基本となるテーブルのデータの場所。次のような値があります。 ● **1** ローカルサーバー ● **3** リモートサーバー
tab_page_list	LONG VARBIT	内部でのみ使用。テーブルの情報を含むページセット (ビットマップで表示されます)。
ext_page_list	LONG VARBIT	内部でのみ使用。テーブルに関するローの拡張と大規模なオブジェクト (LOB) ページを含むページセット (ビットマップで表示されます)。
pct_free	UNSIGNED INT	指定されている場合、テーブルの PCT_FREE の指定、それ以外の場合は NULL。
clustered_index_id	UNSIGNED INT	テーブルのクラスター化されたインデックスの ID。クラスター化されたインデックスがない場合、このフィールドは NULL です。
encrypted	CHAR(1)	テーブルまたはマテリアライズドビューが暗号化されるかどうか。
table_type_str	CHAR(9)	table_type の読み取り可能な値。次のような値があります。 ● **BASE** ベーステーブル ● **MAT VIEW** マテリアライズドビュー ● **GBL TEMP** グローバルテンポラリテーブル ● **VIEW** ビュー

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

基本となるシステムテーブルに関する制約

FOREIGN KEY (dbspace_id) references SYS.ISYSDBSPACE (dbspace_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id)

PRIMARY KEY (table_id)

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

UNIQUE Index (table_name, creator)

参照

- 「SYSVIEW システムビュー」1331 ページ

SYSTABCOL システムビュー

SYSTABCOL システムビューには、データベースの各テーブルとビューの各カラムのローが含まれます。このビューの基本となるシステムテーブルは ISYSTABCOL です。

カラム名	データ型	説明
table_id	UNSIGNED INT	カラムが属するテーブルまたはビューのオブジェクト ID。
column_id	UNSIGNED INT	カラムの ID。各テーブルについて、カラムの番号は 1 から開始されます。 column_id 値によって、SELECT * が使用された場合の結果セットのカラムの順序が決定されます。また、INSERT 文にカラム名のリストが指定されない場合も、この値によってカラム順が決定されます。
domain_id	SMALLINT	カラムのデータ型を、SYSDOMAIN システムビューにリストされているデータ型番号で示します。
nulls	CHAR(1)	NULL 値をカラムに許可するかどうかを指定します。

カラム名	データ型	説明
width	UNSIGNED INT	文字列カラムでは長さ、数値カラムでは精度、その他のデータ型では格納サイズをバイトで示します。
scale	SMALLINT	NUMERIC または DECIMAL データ型のカラムについて、小数点以下の桁数。文字列のカラムの場合、1 の値は文字長のセマンティックを指定します。0 は、バイト長のセマンティックを指定します。
object_id	UNSIGNED BIGINT	テーブルカラムのオブジェクト ID。
max_identity	BIGINT	AUTOINCREMENT、IDENTITY、または GLOBAL AUTOINCREMENT カラムの場合、カラムの最大値。
column_name	CHAR(128)	カラム名。
"default"	LONG VARCHAR	カラムのデフォルト値。この値を指定した場合、INSERT 文が値を指定しないときのみ使われます。
user_type	SMALLINT	ユーザー定義のデータ型を使用してカラムが定義される場合、データ型。
column_type	CHAR(1)	カラムのタイプ (C=計算済みカラム、R=他のカラム)。
compressed	TINYINT	このカラムを圧縮形式で格納するかどうか。
collect_stats	TINYINT	システムが自動的にカラムの統計情報を収集および更新するかどうか。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
inline_max	SMALLINT	ローに格納する BLOB の最大バイト数。NULL 値は、デフォルトが適用されたこと、またはカラムは文字型またはバイナリ型ではないことを示します。NULL ではない inline_max 値は、CREATE TABLE 文または ALTER TABLE 文を使用してカラムに指定した INLINE 値に対応します。
inline_long	SMALLINT	BLOB サイズが inline_max 値を超えた場合、ローに格納されている BLOB のバイトを複製した数。NULL 値は、デフォルトが適用されたこと、またはカラムは文字型またはバイナリ型ではないことを示します。NULL ではない inline_long 値は、CREATE TABLE 文または ALTER TABLE 文を使用してカラムに指定した PREFIX 値に対応します。
lob_index	TINYINT	内部的なしきい値サイズ (約 8 データベースページ) を超過するカラムの BLOB 値に関するインデックスを構築するかどうか。NULL 値は、デフォルトを適用するか、カラムが BLOB タイプであることを示します。1 の値は、インデックスを構築することを示します。0 の値は、インデックスを構築しないことを示します。NULL ではない lob_index 値は、CREATE TABLE 文または ALTER TABLE 文を使用してカラムに指定した INDEX または NO INDEX 値に対応します。

カラム名	データ型	説明
base_type_str	VARCHAR(32,767)	カラムの物理的な型を表す注釈付きの型文字列。

基本となるシステムテーブルに関する制約

PRIMARY KEY (table_id, column_id)

FOREIGN KEY (table_id) references SYS.ISYSTAB (table_id)

FOREIGN KEY (domain_id) references SYS.ISYSDOMAIN (domain_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (user_type) references SYS.ISYSUSERTYPE (type_id)

参照

- 「CREATE TABLE 文」651 ページ

SYSTABLEPERM システムビュー

GRANT 文によってパーミッションが与えられると、SYSTABLEPERM システムビューに格納されます。このビューの各ローは、1 つのテーブル、パーミッションを与えるユーザー ID (grantor)、そしてパーミッションを与えられるユーザー ID (grantee) に対応します。このビューの基本となるシステムテーブルは ISYSTABLEPERM です。

カラム名	データ型	説明
stable_id	UNSIGNED INT	パーミッションが適用されるテーブルまたはビューのテーブル番号。
grantee	UNSIGNED INT	パーミッションを受けるユーザー ID のユーザー番号。
grantor	UNSIGNED INT	パーミッションを付与するユーザー ID のユーザー番号。
selectauth	CHAR(1)	SELECT パーミッションが付与されたかどうかを示します。可能な値は Y、N、または G です。これらの値の詳細については、以下の備考部分を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
insertauth	CHAR(1)	INSERT パーミッションが付与されたかどうかを示します。可能な値は Y、N、または G です。これらの値の詳細については、以下の備考部分を参照してください。
deleteauth	CHAR(1)	DELETE パーミッションが付与されたかどうかを示します。可能な値は Y、N、または G です。これらの値の詳細については、以下の備考部分を参照してください。
updateauth	CHAR(1)	UPDATE パーミッションが、テーブル内のすべてのカラムに付与されたかどうかを示します。可能な値は Y、N、または G です。これらの値の詳細については、以下の備考部分を参照してください。
updatecols	CHAR(1)	UPDATE パーミッションが、基本となるテーブル内の一部のカラムだけに付与されたかどうかを示します。updatecols が "Y" であれば、カラムに UPDATE パーミッションを与える 1 つまたは複数のローが、SYSCOLPERM システムビューにあります。
alterauth	CHAR(1)	ALTER パーミッションが付与されたかどうかを示します。可能な値は Y、N、または G です。これらの値の詳細については、以下の備考部分を参照してください。
referenceauth	CHAR(1)	REFERENCES パーミッションが付与されたかどうかを示します。可能な値は Y、N、または G です。これらの値の詳細については、以下の備考部分を参照してください。

備考

与えられるパーミッションには型がいくつかあります。各パーミッションは、次の 3 つの内の 1 つの値を持ちます。

- **N** No。grantee はこのパーミッションを grantor から付与されていません。
- **Y** Yes。grantee はこのパーミッションを grantor から付与されています。
- **G** grantee はこのパーミッションを受けています。また、grantee は同じパーミッションを他のユーザーに付与できます。

> **注意**
> grantee は同じテーブルに関するパーミッションを、他の grantor から受けることがあります。その場合、この情報は、SYSTABLEPERM システムビューの異なるローで見つかります。

基本となるシステムテーブルに関する制約

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)

参照

- 「GRANT 文」782 ページ

SYSTEXTCONFIG システムビュー

SYSTEXTCONFIG システムビューの各ローは、全文検索機能で使用するテキスト設定オブジェクト 1 つを示します。このビューの基本となるシステムテーブルは ISYSTEXTCONFIG です。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	テキスト設定オブジェクトのオブジェクト ID。
creator	UNSIGNED INT	テキスト設定オブジェクトの作成者。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
term_breaker	TINYINT	文字列を単語に分割するために使用されるアルゴリズム。値は GENERIC の場合は 0、NGRAM の場合は 1 です。GENERIC の場合、英数字以外の文字で区切られた 1 つまたは複数の英数字の文字列は、単語として扱われます。NGRAM は近似一致または単語の区切りにホワイトスペースを使用しないドキュメントに使用します。
stemmer	TINYINT	内部でのみ使用。
min_term_length	TINYINT	1 つの単語に許容される最小文字数。min_term_length より短い単語は無視されます。 MINIMUM TERM LENGTH 設定は、GENERIC 単語区切りの場合にのみ意味を持ちます。NGRAM テキストインデックスの場合、この設定は無視されます。
max_term_length	TINYINT	GENERIC テキストインデックスの場合は、1 つの単語に許容される最大長 (文字数)。max_term_length より長い単語は無視されます。 NGRAM テキストインデックスの場合は、単語が分解される N-gram の長さ。
collation	CHAR(128)	内部でのみ使用。
text_config_name	CHAR(128)	テキスト設定オブジェクトの名前。
prefilter	LONG VARCHAR	外部事前フィルターライブラリの関数名およびライブラリ名です。
postfilter	LONG VARCHAR	内部でのみ使用。

カラム名	データ型	説明
char_stoplist	LONG VARCHAR	CHAR カラムでの全文検索実行時に無視する単語。これらの単語は、テキストインデックスからも省略されます。このカラムは、テキスト設定オブジェクトが default_char から作成されるときに使用されます。
nchar_stoplist	LONG NVARCHAR	NCHAR カラムでの全文検索実行時に無視する単語。これらの単語は、テキストインデックスからも省略されます。このカラムは、テキスト設定オブジェクトが default_nchar から作成されるときに使用されます。
external_term_breaker	LONG VARCHAR	外部単語区切りライブラリの関数名およびライブラリ名です。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

UNIQUE Index (creator, text_config_name)

参照

- 「テキスト設定オブジェクトの作成または変更時の指定項目」『SQL Anywhere サーバー SQL の使用法』
- 「全文検索」『SQL Anywhere サーバー SQL の使用法』

SYSTEXTIDX システムビュー

SYSTEXTIDX システムビューの各ローは、1 つのテキストインデックスを示します。このビューの基本となるシステムテーブルは ISYSTEXTIDX です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
index_id	UNSIGNED BIGINT	SYSIDX のテキストインデックスのオブジェクト ID。
sequence	UNSIGNED INT	内部でのみ使用。
status	UNSIGNED INT	内部でのみ使用。
text_config	UNSIGNED BIGINT	SYSTEXTCONFIG のテキスト設定オブジェクトのオブジェクト ID。
next_handle	UNSIGNED INT	内部でのみ使用。
last_handle	UNSIGNED INT	内部でのみ使用。
deleted_length	UNSIGNED BIGINT	テキストインデックスの削除されたインデックス付きの値の合計サイズ。
pending_length	UNSIGNED BIGINT	次回のリフレッシュ時にテキストインデックスに追加されるインデックス付きの値の合計サイズ。
refresh_type	TINYINT	再表示タイプ。次のいずれかになります。 ● **1** MANUAL ● **2** AUTO ● **3** IMMEDIATE
refresh_interval	UNSIGNED INT	自動リフレッシュの間隔 (分単位)。
last_refresh	TIMESTAMP	最後に行ったリフレッシュの時刻。

基本となるシステムテーブルに関する制約

PRIMARY KEY (index_id, sequence)

FOREIGN KEY (index_id) references SYS.ISYSOBJECT (object_id)

FOREIGN KEY (text_config) references SYS.ISYSTEXTCONFIG (object_id)

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』

SYSTEXTIDXTAB システムビュー

SYSTEXTIDXTAB システムビューの各ローは、テキストインデックスの一部である生成されたテーブルを示します。このビューの基本となるシステムテーブルは ISYSTEXTIDXTAB です。

カラム名	データ型	説明
index_id	UNSIGNED BIGINT	内部でのみ使用。
sequence	UNSIGNED INT	内部でのみ使用。
table_type	UNSIGNED INT	内部でのみ使用。
table_id	UNSIGNED INT	内部でのみ使用。

基本となるシステムテーブルに関する制約

PRIMARY KEY (index_id, sequence, table_type)

FOREIGN KEY (index_id, sequence) references SYS.ISYSTEXTIDX (index_id, sequence)

FOREIGN KEY (table_id) references SYS.ISYSTAB (table_id)

参照

- 「全文検索」『SQL Anywhere サーバー SQL の使用法』

SYSTRIGGER システムビュー

SYSTRIGGER システムビューの各ローは、データベース内のトリガー 1 つを示します。このテーブルには、参照トリガーアクションを持つ外部キー定義に、自動的に作成されるトリガーも含まれます (たとえば、ON DELETE CASCADE)。このビューの基本となるシステムテーブルは ISYSTRIGGER です。

カラム名	データ型	説明
trigger_id	UNSIGNED INT	SYSTRIGGER ビュー内のトリガーのユニークな番号です。
table_id	UNSIGNED INT	トリガーが所属するテーブルのテーブル ID。
object_id	UNSIGNED BIGINT	データベース内のトリガーのオブジェクト ID です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
event	CHAR(1)	トリガーが起動されるきっかけとなる操作。 ● **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)	イベントに関連するトリガーが起動されるタイミング。 ● **A** AFTER (ローレベルのトリガー) ● **B** BEFORE (ローレベルのトリガー) ● **I** INSTEAD OF (ローレベルのトリガー) ● **K** INSTEAD OF (文レベルのトリガー) ● **R** RESOLVE ● **S** AFTER (文レベルのトリガー)
trigger_order	SMALLINT	同じタイプ (insert、update、または delete) の複数のトリガーが、同じ時刻 (BEFORE または AFTER トリガーのみが該当) に起動するように設定されている場合に、トリガーが起動される順序。

カラム名	データ型	説明
foreign_table_id	UNSIGNED INT	参照トリガーアクション (たとえば ON DELETE CASCADE) を持つ外部キー定義のあるテーブルの ID。foreign_table_id 値は、ISYSIDX.table_id の値を反映します。
foreign_key_id	UNSIGNED INT	foreign_table_id によって参照されるテーブルの外部キーの ID。foreign_key_id 値は、ISYSIDX.index_id の値を反映します。
referential_action	CHAR(1)	外部キーによって定義される動作。この 1 文字の値は、外部キーを作成したときに指定した動作に対応します。 ● **C** CASCADE ● **D** SET DEFAULT ● **N** SET NULL ● **R** RESTRICT
trigger_name	CHAR(128)	トリガー名。1 つのテーブルには、同じ名前のトリガーは複数存在できません。
trigger_defn	LONG VARCHAR	トリガーを作成するのに使ったコマンド。
remarks	LONG VARCHAR	トリガーに関する注記。ISYSREMARK システムテーブル内に格納されている値。
source	LONG VARCHAR	トリガーの SQL ソース。この値は、ISYSSOURCE システムテーブル内に格納されています。

基本となるシステムテーブルに関する制約

PRIMARY KEY (trigger_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (table_id) references SYS.ISYSTAB (table_id)

FOREIGN KEY fkey_index (foreign_table_id, foreign_key_id) references SYS.ISYSIDX (table_id, index_id)

UNIQUE Index (table_id, event, trigger_time, trigger_order)

UNIQUE Index (trigger_name, table_id)

UNIQUE Index (table_id, foreign_table_id, foreign_key_id, event)

SYSTYPEMAP システムビュー

SYSTYPEMAP システムビューには、SYSSQLSERVERTYPE システムビューの互換性マッピング値があります。このビューの基本となるシステムテーブルは ISYSTYPEMAP です。

カラム名	データ型	説明
ss_user_type	SMALLINT	Adaptive Server Enterprise ユーザー型を格納します。
sa_domain_id	SMALLINT	対応する SQL Anywhere domain_id を格納します。
sa_user_type	SMALLINT	対応する SQL Anywhere ユーザー型を格納します。
nullable	CHAR(1)	型が NULL 値を許容するかどうか。

基本となるシステムテーブルに関する制約

FOREIGN KEY (sa_domain_id) references SYS.ISYSDOMAIN (domain_id)

SYSUNITOFMEASURE システムビュー

SYSUNITOFMEASURE システムビューの各ローは、データベースに定義されている測定単位に関する記述です。SYSUNITOFMEASURE システムビューの基礎となるテーブルは、ISYSUNITOFMEASURE システムテーブルです。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	システムでのみ使用。
owner	UNSIGNED INT	測定単位の所有者。
unit_name	CHAR(128)	測定単位の名前。

カラム名	データ型	説明
unit_type	CHAR(7)	角度または線形。
conversion_factor	DOUBLE	測定単位の換算係数。

基本となるシステムテーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id)

FOREIGN KEY (owner) references SYS.ISYSUSER (user_id)

UNIQUE constraint (unit_name)

SYSUSER システムビュー

SYSUSER システムビューの各ローは、システム内のユーザーを示します。このビューの基本となるシステムテーブルは ISYSUSER です。

カラム名	データ型	説明
user_id	UNSIGNED INT	ログインポリシーが割り当てられたユーザーのユニークな識別子。
object_id	UNSIGNED BIGINT	データベースのユーザーのユニークな識別子。
user_name	CHAR(128)	ユーザーのログイン名。
password	BINARY(128)	ユーザーのパスワード。
login_policy_id	UNSIGNED BIGINT	ログインポリシーのユニークな識別子。
expired_password_on_login	TINYINT	次回のログイン時にユーザーのパスワードの有効期限が切れるかどうかを示します。
password_creation_time	TIMESTAMP	ユーザーのパスワードが作成された時刻。
failed_login_attempts	UNSIGNED INT	アカウントがロックされるまでにユーザーがログインを失敗できる回数。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
last_login_time	TIMESTAMP	ユーザーが最後にログインした時刻。

注意
SQL Anywhere 12 を使用して作成されたデータベースでは、不正なアクセスからデータを保護するため、このビューの基本となるシステムテーブルは常に暗号化されます。

基本となるシステムテーブルに関する制約

PRIMARY KEY (user_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (login_policy_id) references SYS.ISYSLOGINPOLICY (login_policy_id)

UNIQUE Index (user_name)

参照

- 「SYSLOGINPOLICY システムビュー」1279 ページ
- 「SYSLOGINPOLICYOPTION システムビュー」1280 ページ

SYSUSERAUTHORITY システムビュー

SYSUSERAUTHORITY システムビューの各ローは、1 つのユーザー ID に付与された権限を示します。このビューの基本となるシステムテーブルは、ISYSUSERAUTHORITY です。

カラム名	データ型	説明
user_id	UNSIGNED INT	権限が所属するユーザーの ID。
auth	VARCHAR(20)	ユーザーに付与されている権限。

基本となるシステムテーブルに関する制約

PRIMARY KEY (user_id, auth)

FOREIGN KEY (user_id) references SYS.ISYSUSER (user_id)

SYSUSERMESSAGE システムビュー

SYSUSERMESSAGE システムビューの各ローには、エラーに対するユーザー定義メッセージが格納されまれます。このビューの基本となるシステムテーブルは ISYSUSERMESSAGE です。

注意
SYSUSERMESSAGES システムテーブルに含まれる前のカタログバージョン。このテーブル名は ISYSUSERMESSAGE ('S' なし) に変更され、このビューの基本となるテーブルになります。

カラム名	データ型	説明
error	INTEGER	エラー条件に対するユニークな識別番号。
uid	UNSIGNED INT	メッセージを定義するユーザー番号。
description	VARCHAR(255)	エラー条件に対応するメッセージ。
langid	SMALLINT	予約。

基本となるシステムテーブルに関する制約

FOREIGN KEY (uid) references SYS.ISYSUSER (user_id)

UNIQUE Constraint (error, langid)

SYSUSERTYPE システムビュー

SYSUSERTYPE システムビューの各ローには、ユーザー定義のデータ型の説明が格納されます。このビューの基本となるシステムテーブルは ISYSUSERTYPE です。

カラム名	データ型	説明
type_id	SMALLINT	ユーザー定義データ型のユニークな識別番号。
creator	UNSIGNED INT	データ型の所有者のユーザー番号。
domain_id	SMALLINT	SYSDOMAIN システムビューにリストされたデータ型番号で示す、ユーザー定義データ型の元になるデータ型。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
nulls	CHAR(1)	ユーザー定義データ型が NULL を許可するかどうか。可能な値は Y、N、または U です。値 U は、NULL 入力属性が指定されていないことを示します。
width	BIGINT	文字列カラムでは長さ、数値カラムでは精度、その他のデータ型では記憶領域のサイズをバイト数で示します。
scale	SMALLINT	数値データ型カラムでは小数点以下の桁数、その他のデータ型では 0 を示します。
type_name	CHAR(128)	データ型の名前。
"default"	LONG VARCHAR	データ型のデフォルト値。
"check"	LONG VARCHAR	データ型の CHECK 条件。
base_type_str	VARCHAR(32767)	ユーザータイプの物理的な型を表す注釈付きの型文字列。

基本となるシステムテーブルに関する制約

PRIMARY KEY (type_id)

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

FOREIGN KEY (domain_id) references SYS.ISYSDOMAIN (domain_id)

UNIQUE Constraint (type_name)

SYSVIEW システムビュー

SYSVIEW システムビューの各ローは、データベース内のビューを示します。ビューの追加情報が SYSTAB システムビューにあります。このビューの基本となるシステムテーブルは ISYSVIEW です。

マテリアライズドビューの情報をより読みやすい形式にした sa_materialized_view_info システムプロシージャーも使用できます。

カラム名	データ型	説明
view_object_id	UNSIGNED BIGINT	ビューのオブジェクト ID。
view_def	LONG VARCHAR	ビューの定義 (クエリ仕様)。
mv_build_type	TINYINT	現在使われていません。
mv_refresh_type	TINYINT	ビューに定義された再表示タイプ。可能な値は IMMEDIATE (1) と MANUAL (2) です。
mv_use_in_optimization	TINYINT	クエリの最適化時にマテリアライズドビューを使用できるかどうか (0 = 最適化で使用できません、1 = 最適化で使用できます)。
mv_last_refreshed_at	TIMESTAMP	マテリアライズドビューが最後にリフレッシュされた日時を示します。
mv_known_stale_at	TIMESTAMP	マテリアライズドビューが古くなった時刻。この値は、基本となるベーステーブルのいずれかが、変更され、検証されたときに対応します。0 の値は、ビューが新しいか、古いけれどもデータベースサーバーから古いと印が付けられていないこと (古くなってからそのビューが使用されていない場合など) を示します。sa_materialized_view_info システムプロシージャーを使用して、マテリアライズドビューのステータスを決定します。
mv_last_refreshed_tsn	UNSIGNED BIGINT	マテリアライズドビューをリフレッシュしたトランザクションに割り当てられたシーケンス番号。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

スナップショットアイソレーションによってマテリアライズドビューがリフレッシュされると、mv_last_refreshed_at と mv_last_refreshed_tsn は、マテリアライズドビューの内容の処理中に使用されたローを修正した最初のトランザクションを参照します。

基本となるシステムテーブルに関する制約

PRIMARY KEY (view_object_id)

FOREIGN KEY (view_object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

参照

- 「オプティマイザーによるマテリアライズドビューの使用の有効化または無効化」『SQL Anywhere サーバー SQL の使用法』
- 「再表示タイプの手動または即時への設定」『SQL Anywhere サーバー SQL の使用法』
- 「sa_materialized_view_info システムプロシージャー」1118 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「CREATE MATERIALIZED VIEW 文」576 ページ
- 「REFRESH MATERIALIZED VIEW 文」867 ページ
- 「CREATE VIEW 文」681 ページ

SYSWEBSERVICE システムビュー

SYSWEBSERVICE システムビューの各ローには、Web サービスの説明が格納されます。このビューの基本となるシステムテーブルは ISYSWEBSERVICE です。

カラム名	データ型	説明
service_id	UNSIGNED INT	Web サービスをユニークに識別する番号。
object_id	UNSIGNED BIGINT	Web サービスの ID。
service_name	CHAR(128)	Web サービスに割り当てられた名前。
service_type	VARCHAR(40)	サービスのタイプには、RAW、HTTP、XML、SOAP、DISH などがあります。
auth_required	CHAR(1)	すべての要求に有効なユーザー名とパスワードが必要かどうか。

カラム名	データ型	説明
secure_required	CHAR(1)	HTTP などのセキュリティ保護されていない接続を受け入れるのか、または HTTPS などのセキュリティ保護された接続だけを受け入れるのか。
url_path	CHAR(1)	URL の解釈を制御します。
user_id	UNSIGNED INT	認証が有効の場合、サービス使用のパーミッションを持つユーザーまたはユーザーグループを識別します。認証が無効の場合、要求を処理するときに使用するアカウントを指定します。
parameter	LONG VARCHAR	DISH サービスにインクルードされる SOAP サービスを識別するプレフィクス。
statement	LONG VARCHAR	要求に応答して常に実行される SQL 文。NULL の場合、各要求に含まれる任意の文が代わりに実行されます。DISH 型のサービスでは無視されます。
remarks	LONG VARCHAR	Web サービスに関する注記。ISYSREMARK システムテーブル内に格納されている値。
enabled	CHAR(1)	Web サービスが、現在、有効または無効のいずれであるかを示します (CREATE SERVICE を参照してください)。

基本となるシステムテーブルに関する制約

PRIMARY KEY (service_id)

FOREIGN KEY (object_id) references SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

UNIQUE Constraint (service_name)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

統合ビュー

統合ビューは、ユーザーから柔軟に要求される形式でデータを表示します。たとえば、統合ビューには、一般的に使用されるジョインの機能もあります。統合ビューは、基本となるシステムテーブルの未加工データをそのまま表示するシステムビューとは異なります。たとえば、システムビューのカラムの多くはわかりづらい ID 値ですが、統合ビューは読みやすい名前です。

ST_GEOMETRY_COLUMNS 統合ビュー

ST_GEOMETRY_COLUMNS システムビューの各ローは、データベースに定義されている空間カラムに関する記述です。

カラム名	データ型	説明
table_catalog	VARCHAR(128)	内部で使用されます。
table_schema	CHAR(128)	空間カラムを含むテーブルが属するスキーマの名前。これはテーブル所有者と同じです。
table_name	CHAR(128)	空間カラムを含むテーブルの名前。
column_name	CHAR(128)	空間カラムの名前。
srs_name	CHAR(128)	空間カラムに関連付けられている SRS の名前。SRS がカラムに関連付けられていない場合、srs_name は NULL です。
srs_id	INTEGER	空間カラムに関連付けられている SRS の SRID。
table_id	UNSIGNED INT	カラムが属するテーブルの数値識別子。
column_id	UNSIGNED INT	カラムの数値識別子。
geometry_type_name	VARCHAR(32767)	カラムに含まれているジオメトリの空間データ型 (たとえば、ST_Point、ST_Geometry など)。

参照

- 「空間参照系 (SRS) と空間参照系識別子 (SRID)」『SQL Anywhere サーバー 空間データサポート』
- 「サポートされる空間データ型とその階層」『SQL Anywhere サーバー 空間データサポート』

ST_SPATIAL_REFERENCE_SYSTEMS 統合ビュー

ST_SPATIAL_REFERENCE_SYSTEMS システムビューの各ローは、データベースに定義されている SRS に関する記述です。このビューには、SYSSPATIALREFERENCESYSTEM システムビューとは若干異なる量の情報があります。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	システムでのみ使用。
owner	UNSIGNED INT	SRS の所有者。
srs_name	CHAR(128)	SRS の名前
srs_id	INT	空間参照系の数値識別子 (SRID)。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
srs_type	CHAR(11)	SQL/MM 標準による定義に従った SRS のタイプ。次のいずれかの値を取ります。 ● **GEOGRAPHIC** 緯度と経度 (および標高) の軸を持つ、測地参照される座標系に基づく SRS 用です。これらの SRS は、PLANAR タイプまたは ROUND EARTH タイプです。 ● **PROJECTED** 緯度と経度の軸を持たず、測地参照される座標系に基づく SRS 用です。これらの SRS は PLANAR タイプです。 ● **ENGINEERING** 測地参照されない座標系に基づく SRS 用です。これらの SRS は PLANAR タイプです。 ● **GEOCENTRIC** サポートされていません。 ● **COMPOUND** サポートされていません。 ● **VERTICAL** サポートされていません。 srs_type が空の場合、タイプは指定されません。
round_earth	CHAR(1)	SRS タイプが ROUND EARTH (Y) または PLANAR (N) のいずれであるか。

カラム名	データ型	説明
axis_order	CHAR(12)	データベースサーバーが緯度と経度に関連してポイントを解釈する方法を記述します (たとえば、ST_Lat メソッドと ST_Long メソッドを使用する場合)。非地理的空間参照系の場合、軸順序は x/y/z/m です。地理的空間参照系の場合、デフォルトの軸順序は long/lat/z/m であり、lat/long/z/m もサポートされます。
snap_to_grid	DOUBLE	SQL Anywhere が計算を実行するときに使用するグリッドのサイズを定義します。
tolerance	DOUBLE	ポイントを比較するときに使用する精度を定義します。
semi_major_axis	DOUBLE	ROUND EARTH SRS の楕円の中心から赤道までの距離です。
semi_minor_axis	DOUBLE	ROUND EARTH SRS の楕円の中心から両極までの距離です。
inv_flattening	DOUBLE	ROUND EARTH SRS の楕円に使用される逆扁平率です。これは、次の式によって算出される比率となります。 1/f = (semi-major-axis) / (semi-major-axis - semi-minor-axis)
min_x	DOUBLE	座標において許される x の最小値。
max_x	DOUBLE	座標において許される x の最大値。
min_y	DOUBLE	座標において許される y の最小値。
max_y	DOUBLE	座標において許される y の最大値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カラム名	データ型	説明
min_z	DOUBLE	座標において許される z の最小値。
max_z	DOUBLE	座標において許される z の最大値。
min_m	DOUBLE	座標において許される m の最小値。
max_m	DOUBLE	座標において許される m の最大値。
min_lat	DOUBLE	座標において許される緯度の最小値。
max_lat	DOUBLE	座標において許される緯度の最大値。
min_long	DOUBLE	座標において許される経度の最小値。
max_long	DOUBLE	座標において許される経度の最大値。
organization	LONG VARCHAR	空間参照系によって使用される座標系を作成した組織の名前。
organization_coordsys_id	INT	座標系を作成した組織が座標系に対して指定した ID。
linear_unit_of_measure	CHAR(128)	SRS によって使用される線形測定単位。
angular_unit_of_measure	CHAR(128)	SRS によって使用される角度測定単位。
polygon_format	LONG VARCHAR	多角形のリングの方向。CounterClockwise、ClockWise、または EvenOdd のいずれかです。

カラム名	データ型	説明
storage_format	LONG VARCHAR	データが正規化フォーマット (Internal)、非正規化フォーマット (Original)、または両方 (Mixed) のいずれで格納されるか。
definition	LONG VARCHAR	追加の定義設定。
transform_definition	LONG VARCHAR	この SRS から別の SRS にデータを変換するときに使用される変換定義設定。
description	LONG VARCHAR	SRS の説明。

参照

- 「SYSSPATIALREFERENCESYSTEM システムビュー」1302 ページ
- 「CREATE SPATIAL REFERENCE SYSTEM 文」635 ページ

ST_UNITS_OF_MEASURE 統合ビュー

ST_UNITS_OF_MEASURE システムビューの各ローは、データベースに定義されている測定単位に関する記述です。このビューには、SYSUNITOFMEASURE システムビューよりも多い情報があります。

カラム名	データ型	説明
object_id	UNSIGNED BIGINT	システムでのみ使用。
owner	UNSIGNED INT	測定単位の所有者。
unit_name	CHAR(128)	測定単位の名前。
unit_type	CHAR(7)	角度または線形。
conversion_factor	DOUBLE	測定単位の換算係数。
description	LONG VARCHAR	測定単位の説明。

SYSARTICLECOLS 統合ビュー

SYSARTICLECOLS ビューの各ローは、アーティクル内のカラムを識別します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSARTICLECOL システムビュー」1255 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSTABCOL システムビュー」1315 ページ

SYSARTICLES 統合ビュー

SYSARTICLES ビューの各ローは、パブリケーション内のアーティクルを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSARTICLE システムビュー」1254 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSCAPABILITIES 統合ビュー

SYSCAPABILITIES ビューの各ローは、リモートデータベースサーバーの機能のステータスを示します。このビューは、ISYSCAPABILITY と ISYSCAPABILITYNAME の各システムテーブルからデータを取得します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSCAPABILITIES"
  as select
ISYSCAPABILITY.capid,ISYSCAPABILITY.srvid,property('RemoteCapability',ISYSCAPABILITY.capid)
as capname,ISYSCAPABILITY.capvalue
    from SYS.ISYSCAPABILITY
```

参照

- 「SYSCAPABILITY システムビュー」1255 ページ
- 「SYSCAPABILITYNAME システムビュー」1256 ページ

SYSCATALOG 統合ビュー

SYSCATALOG ビューの各ローは、システムテーブルを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSTAB システムビュー」1311 ページ
- 「SYSTABCOL システムビュー」1315 ページ
- 「ISYSDBSPACE システムテーブル」996 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSIDX システムビュー」1273 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSCOLAUTH 統合ビュー

SYSCOLAUTH ビューの各ローは、カラムに付与されている一連の権限 (UPDATE、SELECT、または REFERENCES) を示します。SYSCOLAUTH ビューは、「SYSCOLPERM システムビュー」1257 ページをユーザーがわかりやすい表示形式にしたデータです。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSCOLPERM システムビュー」1257 ページ
- 「SYSTABCOL システムビュー」1315 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSTAB システムビュー」1311 ページ

SYSCOLSTATS 統合ビュー

SYSCOLSTATS ビューには、オプティマイザーによって使用されるカラム統計が、ヒストグラムとして格納されます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSCOLSTATS"
  as select u.user_name,t.table_name,c.column_name,
    s.format_id,s.update_time,s.density,s.max_steps,
    s.actual_steps,s.step_values,s.frequencies
    from SYS.ISYSCOLSTAT as s
      join SYS.ISYSTABCOL as c on(s.table_id = c.table_id
      and s.column_id = c.column_id)
      join SYS.ISYSTAB as t on(t.table_id = c.table_id)
      join SYS.ISYSUSER as u on(u.user_id = t.creator)
```

参照

- 「SYSCOLSTAT システムビュー」1258 ページ
- 「SYSTABCOL システムビュー」1315 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSCOLUMNS 統合ビュー

SYSCOLUMNS ビューの各ローは、カタログ内の各テーブルとビューのカラム 1 つを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSTABCOL システムビュー」1315 ページ
- 「SYSIDXCOL システムビュー」1275 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSDOMAIN システムビュー」1262 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSFOREIGNKEYS 統合ビュー

SYSFOREIGNKEYS ビューの各ローは、カタログ内の各テーブルの外部キー 1 つを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSTAB システムビュー」1311 ページ
- 「SYSTABCOL システムビュー」1315 ページ
- 「ISYSIDX システムテーブル」997 ページ
- 「SYSIDXCOL システムビュー」1275 ページ
- 「SYSFKEY システムビュー」1268 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSDOMAIN システムビュー」1262 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSGROUPS 統合ビュー

SYSGROUPS ビューには、各グループの各メンバーに 1 つのローがあります。このビューは、グループとメンバーの多対多の関係を示します。グループには複数のメンバーがあり、あるユーザーは複数のグループのメンバーになれます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSGROUPS"( group_name,
  member_name )
  as select g.user_name,u.user_name
    from SYS.ISYSGROUP,SYS.ISYSUSER as g,SYS.ISYSUSER as u
    where ISYSGROUP.group_id = g.user_id and ISYSGROUP.group_member = u.user_id
```

参照

- 「SYSUSER システムビュー」1328 ページ
- 「SYSGROUP システムビュー」1269 ページ

SYSINDEXES 統合ビュー

SYSINDEXES ビューの各ローは、データベース内のインデックス 1 つを示します。このビューの代わりに、SYSIDX と SYSIDXCOL のシステムビューを使用することもできます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSIDX システムビュー」1273 ページ
- 「SYSTABCOL システムビュー」1315 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSDBSPACE システムビュー」1260 ページ
- 「SYSIDXCOL システムビュー」1275 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSOPTIONS 統合ビュー

SYSOPTIONS ビューの各ローは、SET コマンドを使用して作成されているオプション 1 つを示します。各ユーザーはオプションごとに自分の設定を保存できます。また、ユーザー ID PUBLIC に対する設定は、自分の設定を持たないユーザーが使うデフォルトの設定になります。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSOPTION システムビュー」1285 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSPROCAUTH 統合ビュー

SYSPROCAUTH ビューの各ローは、プロシージャーに付与されている一連の権限を示します。または、SYSPROCPERM システムビューを使用することもできます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSPROCEDURE システムビュー」1286 ページ
- 「SYSPROCPERM システムビュー」1290 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSPROCPARMS 統合ビュー

SYSPROCPARMS ビューの各ローは、データベース内のプロシージャーに対するパラメーターを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSPROCPARM システムビュー」1288 ページ
- 「SYSPROCEDURE システムビュー」1286 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSDOMAIN システムビュー」1262 ページ
- 「SYSUSERTYPE システムビュー」1330 ページ

SYSPROCS 統合ビュー

SYSPROCS ビューには、プロシージャーまたはファンクションに記録されているプロシージャー名、ファンクション名、作成者名、コメントが表示されます。

ビューを構成するテーブルとカラムは、以下の ALTER VIEW 文で示されます。

```
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)
```

参照

- 「SYSPROCEDURE システムビュー」1286 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSPUBLICATIONS 統合ビュー

SYSPUBLICATIONS ビューの各ローは、SQL Remote または Mobile Link パブリケーションを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSPUBLICATION システムビュー」1291 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSREMOTEOPTION2 統合ビュー

機密データを含まない SYSREMOTEOPTION と SYSREMOTEOPTIONTYPE のカラムを、読み易い形式で表示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSREMOTEOPTION2"
  as select ISYSREMOTEOPTION.option_id,
```

```
ISYSREMOTEOPTION.user_id,
SYS.HIDE_FROM_NON_DBA(ISYSREMOTEOPTION.setting) as setting
from SYS.ISYSREMOTEOPTION
```

参照

- 「SYSREMOTEOPTION システムビュー」1293 ページ

SYSREMOTEOPTIONS 統合ビュー

SYSREMOTEOPTIONS ビューの各ローは、SQL Remote メッセージリンクパラメーターの値を示します。このビューの一部のカラムには、機密データが含まれている可能性があります。このため、このビューにアクセスできるのは DBA 権限を持つユーザーに限られます。SYSREMOTEOPTION2 ビューを使用すると、機密データ以外のデータにパブリックアクセスできます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSREMOTETYPE システムビュー」1294 ページ
- 「SYSREMOTEOPTIONTYPE システムビュー」1294 ページ
- 「SYSREMOTEOPTION システムビュー」1293 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSREMOTETYPES 統合ビュー

SYSREMOTETYPES ビューの各ローは、パブリッシャーアドレスなど、SQL Remote のメッセージタイプ 1 つを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSREMOTETYPE システムビュー」1294 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSREMOTEUSERS 統合ビュー

SYSREMOTEUSERS ビューの各ローは、REMOTE パーミッションを持つユーザー ID (サブスクライバー) を示します。そのユーザーに送信された SQL Remote メッセージと、そのユーザーから送信されたメッセージのステータスも合わせて示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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(a.time_sent,60*hour(a.send_time)
        +minute(seconds(a.send_time,59))))
        from SYS.ISYSREMOTEUSER as a where a.frequency = 'P'
        and a.send_time = r.send_time)
     endif
   else if current date+r.send_time
      > coalesce(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,r.time_sent,r.log_sent,r.confirm_sent,r.send_count,
  r.resend_count,r.time_received,r.log_received,
  r.confirm_received,r.receive_count,r.rereceive_count
  from SYS.ISYSREMOTEUSER as r
   join SYS.ISYSUSER as u on(u.user_id = r.user_id)
   join SYS.ISYSREMOTETYPE as t on(t.type_id = r.type_id)
```

参照

- 「SYSREMOTEUSER システムビュー」1295 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSREMOTETYPE システムビュー」1294 ページ

SYSSUBSCRIPTIONS 統合ビュー

REMOTE パーミッションを持つあるユーザー ID からの、あるパブリケーションに対するサブスクリプションについて、各ローで示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSSUBSCRIPTION システムビュー」1307 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSSYNC2 統合ビュー

SYSSYNC2 ビューから、機密情報を公開することなく、SYSSYNC システムビューで見つかったデータ (Mobile Link 同期に関する情報) にパブリックアクセスできます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSSYNC2"
  as select ISYSSYNC.sync_id,
    ISYSSYNC.type,
    ISYSSYNC.publication_id,
    ISYSSYNC.progress,
    ISYSSYNC.site_name,
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) 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
    from SYS.ISYSSYNC
```

参照

- 「SYSSYNC システムビュー」1308 ページ

SYSSYNCPUBLICATIONDEFAULTS 統合ビュー

SYSSYNCPUBLICATIONDEFAULTS は、Mobile Link 同期に関連するパブリケーションに対応したデフォルトの同期設定のビューです。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSSYNCPUBLICATIONDEFAULTS"
  as select s.sync_id,
    p.publication_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
```

```
    s.server_conn_type
    from SYS.ISYSSYNC as s join SYS.ISYSPUBLICATION as p on(p.publication_id = s.publication_id)
where
    s.site_name is null
```

参照

- 「SYSSYNC システムビュー」1308 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ

SYSSYNCS 統合ビュー

SYSSYNCS ビューには、Mobile Link 同期に関する情報が入っています。このビューの一部のカラムには、機密データが含まれている可能性があります。このため、このビューにアクセスできるのは DBA 権限を持つユーザーに限られます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSSYNCS"
  as select p.publication_name,s.progress,s.site_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as 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.ISYSSYNC as s
      left outer join SYS.ISYSPUBLICATION as p
      on p.publication_id = s.publication_id
```

参照

- 「SYSSYNC システムビュー」1308 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ

SYSSYNCSCRIPTS 統合ビュー

SYSSYNCSCRIPTS ビューの各ローは、Mobile Link のスクリプト化されたアップロードに関するストアドプロシージャーを識別します。このビューは、SYSSYNCSCRIPT システムビューとほとんど同じですが、データが未加工形式ではなく、ユーザーが読みやすい形式で表示される点が異なります。

```
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
```

参照

- 「SYSSYNCSCRIPT システムビュー」1310 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSPROCEDURE システムビュー」1286 ページ
- 「スクリプト化されたアップロード」『Mobile Link クライアント管理』

SYSSYNCSUBSCRIPTIONS 統合ビュー

SYSSYNCSUBSCRIPTIONS ビューには、Mobile Link 同期のサブスクリプションと関連付けられた同期設定が含まれます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSSYNCSUBSCRIPTIONS"
  as select s.sync_id,
    p.publication_name,
    s.progress,
    s.site_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as 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.ISYSSYNC as s join SYS.ISYSPUBLICATION 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)
```

参照

- 「SYSSYNC システムビュー」1308 ページ
- 「SYSPUBLICATION システムビュー」1291 ページ
- 「SYSSYNCUSERS 統合ビュー」1353 ページ

SYSSYNCUSERS 統合ビュー

Mobile Link 同期ユーザーに関連する同期設定のビューです。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSSYNCUSERS"
  as select ISYSSYNC.sync_id,
    ISYSSYNC.site_name,
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) as server_connect,
    ISYSSYNC.server_conn_type
    from SYS.ISYSSYNC where
    ISYSSYNC.publication_id is null
```

参照

- 「SYSSYNC システムビュー」1308 ページ

SYSTABAUTH 統合ビュー

SYSTABAUTH ビューには、SYSTABLEPERM システムビューの情報が表示されますが、より読みやすい形式です。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSTABAUTH"( grantor,
  grantee,screator,stname,tcreator,ttname,
  selectauth,insertauth,deleteauth,
  updateauth,updatecols,alterauth,referenceauth )
  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
    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
```

参照

- 「SYSTABLEPERM システムビュー」1318 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSTAB システムビュー」1311 ページ

SYSTRIGGERS 統合ビュー

SYSTRIGGERS ビューの各ローは、データベース内のトリガー 1 つを示します。このテーブルには、参照トリガーアクションを持つ外部キー定義に、自動的に作成されるトリガーも含まれます (たとえば、ON DELETE CASCADE)。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSTRIGGER システムビュー」1324 ページ
- 「SYSTAB システムビュー」1311 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSUSEROPTIONS 統合ビュー

SYSUSEROPTIONS ビューには、各ユーザーに有効なオプション設定が含まれます。ユーザーにオプション設定がない場合、このビューには PUBLIC のオプション設定が表示されます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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'
```

参照

- 「SYSOPTIONS 統合ビュー」1346 ページ
- 「SYSUSER システムビュー」1328 ページ

SYSVIEWS 統合ビュー

SYSVIEWS ビューの各ローには、ビュー定義など、ビューに関して記述します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSTAB システムビュー」1311 ページ
- 「SYSVIEW システムビュー」1331 ページ
- 「SYSUSER システムビュー」1328 ページ

互換ビュー

互換ビューは、SQL Anywhere 10 以前のバージョンとの互換性のために提供されるビューです。将来的なリリースで互換ビューを減らす可能性があるため、可能な場合、システムビューと統合ビューを使用することをおすすめします。

SYSCOLLATION 互換ビュー (旧式)

SYSCOLLATION 互換ビューには、データベースの照合順情報が格納されます。組み込み関数経由で取得でき、カタログには保存されません。このビューの定義を次に示します。

```
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
```

参照

- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「DB_PROPERTY 関数 [システム]」208 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」203 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SYSCOLLATIONMAPPINGS 互換ビュー (旧式)

SYSCOLLATIONMAPPINGS 互換ビューにはデータベース照合マッピングを持つローが 1 つだけ含まれています。組み込み関数経由で取得でき、カタログには保存されません。このビューの定義を次に示します。

```
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
```

参照

- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「DB_PROPERTY 関数 [システム]」208 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」203 ページ

SYSCOLUMN 互換ビュー (旧式)

SYSCOLUMN ビューは、SYSCOLUMN システムテーブルを提供していた古いバージョンのソフトウェアとの互換性を保つために用意されています。ただし、以前の SYSCOLUMN テーブルは、SYSTABCOL システムビューに対応する ISYSTABCOL システムテーブルで置換されたため、ISYSTABCOL の使用をおすすめします。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSTABCOL システムビュー」1315 ページ
- 「SYSREMARK システムビュー」1293 ページ
- 「SYSIDXCOL システムビュー」1275 ページ

SYSFILE 互換ビュー (旧式)

SYSFILE システムビューの各ローは、データベースの DB 領域を示します。各データベースは、1 つ以上の DB 領域で構成されます。各 DB 領域は、1 つのオペレーティングシステムファイルに対応します。

メインデータベースファイル、テンポラリファイル、トランザクションログファイル、トランザクションログミラーファイル用に、自動的に DB 領域が作成されます。トランザクションログ、トランザクションログミラーの DB 領域に関する情報は、SYSFILE システムビューには表示されません。

```
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)
```

参照

- 「事前定義の DB 領域」『SQL Anywhere サーバー データベース管理』

SYSFKCOL 互換ビュー (旧式)

SYSFKCOL の各ローは、外部テーブルの外部カラムと、プライマリテーブルのプライマリカラムの関係を示します。このビューは使用されなくなりました。代わりに SYSIDX と SYSIDXCOL のシステムビューを使用してください。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SYSIDX システムビュー」1273 ページ
- 「SYSIDXCOL システムビュー」1275 ページ

SYSFOREIGNKEY 互換ビュー (旧式)

SYSFOREIGNKEY ビューは、SYSFOREIGNKEY システムテーブルを提供していた古いバージョンのソフトウェアとの互換性を保つために用意されています。ただし、以前の SYSFOREIGNKEY システムテーブルは、SYSFKEY システムビューに対応する ISYSFKEY システムテーブルで置換されたため、ISYSFKEY の使用をおすすめします。

外部キーは、外部テーブルとプライマリテーブルの 2 つのテーブル間の関係です。外部キーは、SYSFOREIGNKEY の 1 つのローと SYSFKCOL の 1 つまたは複数のローで定義されます。SYSFOREIGNKEY には、外部キーに関する一般的な情報が入っています。SYSFKCOL は外部キーのカラムを識別して、外部キーの各カラムとプライマリテーブルの中のプライマリキーのカラムを関連付けます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSIDX システムビュー」1273 ページ
- 「SYSPHYSIDX システムビュー」1285 ページ
- 「SYSREMARK システムビュー」1293 ページ
- 「SYSFKEY システムビュー」1268 ページ

SYSINDEX 互換ビュー (旧式)

SYSINDEX ビューは、SYSINDEX システムテーブルを提供していた古いバージョンのソフトウェアとの互換性を保つために用意されています。ただし、以前の SYSINDEX テーブルは、

SYSIDX システムビューに対応する ISYSIDX システムテーブルで置換されたため、ISYSIDX の使用をおすすめします。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSIDX システムビュー」1273 ページ
- 「SYSPHYSIDX システムビュー」1285 ページ
- 「SYSTABLE 互換ビュー (旧式)」1361 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSINFO 互換ビュー (旧式)

SYSINFO ビューは、データベースが作成されたときに定義された、データベースの情報を示します。このテーブルは常に 1 つのローを持ちます。このビューは組み込み関数経由で取得でき、カタログには保存されません。次に SYSINFO ビューの定義を示します。

```
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
```

参照

- 「データベースプロパティ値のアクセス」『SQL Anywhere サーバー データベース管理』
- 「DB_PROPERTY 関数 [システム]」208 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」203 ページ

SYSIXCOL 互換ビュー (旧式)

SYSIXCOL ビューは、SYSIXCOL システムテーブルを提供していた古いバージョンのソフトウェアとの互換性を保つために用意されています。ただし、SYSIXCOL システムテーブルは ISYSIDXCOL システムテーブルで置換され、ISYSIDXCOL システムビューに対応しています。SYSIDXCOL システムビューの使用をおすすめします。

SYSIXCOL の各ローは、インデックスのカラムを示します。ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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
```

参照

- 「SYSIDX システムビュー」1273 ページ
- 「SYSIDXCOL システムビュー」1275 ページ

SYSTABLE 互換ビュー (旧式)

SYSTABLE ビューは、SYSTABLE システムテーブルを提供していた古いバージョンのソフトウェアとの互換性を保つために用意されています。ただし、以前の SYSTABLE テーブルは、SYSTAB システムビューに対応する ISYSTAB システムテーブルで置換されたため、SYSTAB の使用をおすすめします。

SYSTABLE ビューの各ローは、データベース内のテーブル 1 つを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
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)
```

参照

- 「SYSTAB システムビュー」1311 ページ
- 「SYSREMARK システムビュー」1293 ページ
- 「SYSSOURCE システムビュー」1301 ページ
- 「SYSVIEW システムビュー」1331 ページ
- 「SYSPROXYTAB システムビュー」1291 ページ
- 「SYSIDX システムビュー」1273 ページ
- 「SYSPHYSIDX システムビュー」1285 ページ

SYSUSERAUTH 互換ビュー (旧式)

SYSUSERAUTH ビューは、古いバージョンのソフトウェアとの互換性を保つために用意されています。代わりに SYSUSERAUTHORITY システムビューを使用してください。

SYSUSERAUTH ビューの各ローは、user_id を公開せずに、ユーザーに関して記述します。各ユーザーは、ユーザー名で識別されます。このビューにはパスワードが表示されるため、PUBLIC SELECT パーミッションはありません。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSUSERAUTH"( name,
  password,resourceauth,dbaauth,scheduleauth,user_group )
  as select
SYSUSERPERM.user_name,SYSUSERPERM.password,SYSUSERPERM.resourceauth,SYSUSERPE
RM.dbaauth,SYSUSERPERM.scheduleauth,SYSUSERPERM.user_group
    from SYS.SYSUSERPERM
```

参照

- 「SYSUSERAUTHORITY システムビュー」1329 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」1363 ページ

SYSUSERLIST 互換ビュー (旧式)

SYSUSERAUTH ビューは、古いバージョンのソフトウェアとの互換性を保つために用意されています。

SYSUSERLIST ビューの各ローは、user_id とパスワードを公開せずに、ユーザーに関して記述します。各ユーザーは、ユーザー名で識別されます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSUSERLIST"( name,
  resourceauth,dbaauth,scheduleauth,user_group )
  as select
SYSUSERPERM.user_name,SYSUSERPERM.resourceauth,SYSUSERPERM.dbaauth,SYSUSERPER
M.scheduleauth,SYSUSERPERM.user_group
    from SYS.SYSUSERPERM
```

参照

- 「SYSUSERPERM 互換ビュー (旧式)」1363 ページ

SYSUSERPERM 互換ビュー (旧式)

前のバージョンで使用できる権限とパーミッションが表示されるだけなので、このビューは使用されなくなりました。アプリケーションを変更して代わりに SYSUSERAUTHORITY システムビューを使用してください。

SYSUSERPERM ビューの各ローは、ユーザー ID 1 つを示します。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSUSERPERM"
  as select b.user_id,
    b.object_id,
```

```
    b.user_name,
    b.password,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth =
'RESOURCE') then
      'Y' else 'N' endif as resourceauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth = 'DBA') then
      'Y' else 'N' endif as dbaauth,
    'N' as scheduleauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth = 'PUBLISH')
then
      'Y' else 'N' endif as publishauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth = 'REMOTE
DBA') then
      'Y' else 'N' endif as remotedbaauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth = 'GROUP')
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)
```

参照

- 「SYSUSERAUTHORITY システムビュー」1329 ページ
- 「SYSUSER システムビュー」1328 ページ
- 「SYSREMARK システムビュー」1293 ページ

SYSUSERPERMS 互換ビュー (旧式)

前のバージョンで使用できる権限とパーミッションが表示されるだけなので、このビューは使用されなくなりました。アプリケーションを変更して代わりに SYSUSERAUTHORITY および SYSUSER システムビューを使用してください。

SYSUSERPERMS ビューと同様に、SYSUSERPERMS ビューの各ローは ユーザー ID 1 つを示します。ただし、パスワード情報は含まれません。すべてのユーザーがこのビューを表示できます。

ビューを構成するテーブルとカラムは、以下の SQL 文で示されます。特定のテーブルまたはカラムの詳細については、以下のビュー定義にあるリンクを参照してください。

```
ALTER VIEW "SYS"."SYSUSERPERMS"
  as select
SYSUSERPERM.user_id,SYSUSERPERM.user_name,SYSUSERPERM.resourceauth,SYSUSERPER
M.dbaauth,

SYSUSERPERM.scheduleauth,SYSUSERPERM.user_group,SYSUSERPERM.publishauth,SYSUSER
PERM.remotedbaauth,SYSUSERPERM.remarks
    from SYS.SYSUSERPERM
```

参照

- 「SYSUSERPERM 互換ビュー (旧式)」1363 ページ
- 「SYSUSERAUTHORITY システムビュー」1329 ページ
- 「SYSUSER システムビュー」1328 ページ

Transact-SQL 互換のビュー

Adaptive Server Enterprise と SQL Anywhere のシステムカタログは異なります。Adaptive Server Enterprise システムテーブルとビューは、ユーザー dbo が所有しています。一部は master データベースにあり、一部は sybsecurity データベースにあり、一部は個々のデータベースにあります。SQL Anywhere システムのテーブルとビューは、SYS という特殊なユーザーが所有し、各データベースに別々に存在しています。

互換アプリケーションを使用しやすいように、SQL Anywhere は特殊なユーザー dbo が所有する次のビューセットを用意しています。これは Adaptive Server Enterprise のビューに対応しています。構造上の違いのために、特定の Adaptive Server Enterprise テーブルまたはビューの内容が SQL Anywhere のコンテキストで無意味になる場合、そのビューは空であり、カラム名とデータ型だけを持っています。

ビュー名	説明
syscolumns	テーブルまたはビューのカラムごとに、またプロシージャーのパラメーターごとに 1 ロー追加
syscomments	ビュー、ルール、デフォルト、トリガー、プロシージャーごとに 1 つ以上のローを追加。SQL 定義文を表示。
sysindexes	クラスタードインデックスまたはノンクラスタードインデックスごとに 1 ロー、インデックスのないテーブルごとに 1 ロー、および text または image データを持つテーブルごとに 1 ロー追加
sysobjects	テーブル、ビュー、プロシージャー、ルール、トリガー、デフォルト、ログ、またはテンポラリオブジェクト (tempdb 内でのみ) ごとに 1 ロー追加
systypes	システム提供またはユーザー定義のデータ型ごとに 1 ロー追加
sysusers	データベースについての許可を持つユーザーごとに 1 ロー追加

ビュー名	説明
syslogins	有効なユーザーアカウントごとに 1 ロー追加

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

索引

記号

// コメントインジケーター
説明, 84
/* コメントインジケーター
説明, 84
^
ビット処理演算子, 11
~
ビット処理演算子, 11
@@char_convert グローバル変数
説明, 79
@@client_csid グローバル変数
説明, 79
@@client_csname グローバル変数
説明, 79
@@connections グローバル変数
説明, 79
@@cpu_busy グローバル変数
説明, 79
@@dbts グローバル変数
説明, 79
@@error グローバル変数
説明, 79
@@fetch_status グローバル変数
説明, 79
@@identity グローバル変数
説明, 79, 83
トリガー, 83
@@idle グローバル変数
説明, 79
@@io_busy グローバル変数
説明, 79
@@isolation グローバル変数
説明, 79
@@langid グローバル変数
説明, 79
@@language グローバル変数
説明, 79
@@max_connections グローバル変数
説明, 79
@@maxcharlen グローバル変数
説明, 79
@@ncharsize グローバル変数
説明, 79
@@nestlevel グローバル変数
説明, 79
@@pack_received グローバル変数
説明, 79
@@pack_sent グローバル変数
説明, 79
@@packet_errors グローバル変数
説明, 79
@@procid グローバル変数
説明, 79
@@rowcount グローバル変数
説明, 79
@@servername グローバル変数
説明, 79
@@spid グローバル変数
説明, 79
@@sqlstatus グローバル変数
説明, 79
@@textsize グローバル変数
説明, 79
@@thresh_hysteresis グローバル変数
説明, 79
@@timeticks グローバル変数
説明, 79
@@total_errors グローバル変数
説明, 79
@@total_read グローバル変数
説明, 79
@@total_write グローバル変数
説明, 79
@@tranchained グローバル変数
説明, 79
@@trancount グローバル変数
説明, 79
@@transtate グローバル変数
説明, 79
@@version グローバル変数
トリガー, 79
@HttpMethod
HTTP_HEADER 関数, 250
@HttpQueryString
HTTP_HEADER 関数, 250
@HttpStatus ヘッダー
HTTP_RESPONSE_HEADER 関数, 252
sa_set_http_header_info システムプロシージャー, 1185
@HttpURI
HTTP_HEADER 関数, 250

@HttpVersion
HTTP_HEADER 関数, 250
@mp:id メタプロパティ
openxml システムプロシージャー, 1036
openxml システムプロシージャー, 1036
openxml システムプロシージャー, 1036
openxml システムプロシージャー, 1036
openxml システムプロシージャー, 1036
&
ビット処理演算子, 11
% 演算子
モジュロ関数, 282
% コメントインジケーター
説明, 84
|
ビット処理演算子, 11
0x
バイナリリテラル, 6
16 進
16 進値との変換, 6
16 進
CAST、CONVERT、HEXTOINT、INTTOHEX 関数を使用した変換, 6
16 進定数
16 進値との変換, 6
バイナリとして処理, 6
16 進のエスケープシーケンス
SQL 文字列内, 7
16 進文字列
説明, 241
2 月 29 日
説明, 111
3 値的論理
NULL 値, 68
構文, 59
- コメントインジケーター
説明, 84

A

BEFORE トリガー
CREATE TRIGGER 文, 670
ABSOLUTE 句
FETCH 文, 750
ABS 関数
構文, 151
AccentSensitive プロパティ
DB_EXTENDED_PROPERTY 関数, 203
ACCENT 句
CREATE DATABASE 文, 522
AcceptCharset オプション
sa_set_http_option システムプロシージャー, 1186
Accept-Encoding
HTTP_HEADER 関数, 249
ACOS 関数
構文, 152
Adaptive Server Enterprise
CREATE DATABASE 文, 522
sa_migrate システムプロシージャーを使用して SQL Anywhere にマイグレート, 1128
Watcom SQL 構文へのストアドプロシージャーの変換, 388
システムプロシージャー, 1033
ADD | ALTER | DELETE SCHEDULE 句
ALTER EVENT 文, 418
ADD OPTION 句
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
ALTER SYNCHRONIZATION USER 文 [Mobile Link], 461
ADD PCTFREE 句
ALTER MATERIALIZED VIEW 文, 427
ADDRESS 句
ALTER REMOTE MESSAGE TYPE 文, 436
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
ALTER SYNCHRONIZATION USER 文 [Mobile Link], 461
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 613
CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 648
CREATE SYNCHRONIZATION USER, 650
GRANT CONSOLIDATE 文 [SQL Remote], 776
GRANT REMOTE 文 [SQL Remote], 780
ADD table-constraint 句
ALTER TABLE 文, 469
ADD 句
ALTER DBSPACE 文, 414
ALTER PUBLICATION 文, 435
ALTER TABLE 文, 464
AES_FIPS 暗号化アルゴリズム
CREATE DATABASE 文, 521
CREATE ENCRYPTED FILE 文, 537
DECRYPT 関数, 211

ENCRYPT 関数, 216
AES256_FIPS 暗号化アルゴリズム
CREATE DATABASE 文, 521
CREATE ENCRYPTED FILE 文, 537
DECRYPT 関数, 211
ENCRYPT 関数, 216
AES256 暗号化アルゴリズム
CREATE DATABASE 文, 521
CREATE ENCRYPTED FILE 文, 537
DECRYPT 関数, 211
ENCRYPT 関数, 216
AES 暗号化アルゴリズム
CREATE DATABASE 文, 521
CREATE ENCRYPTED FILE 文, 537
DECRYPT 関数, 211
ENCRYPT 関数, 216
AFTER MESSAGE BREAK 句
WAITFOR 文, 986
AFTER 句
CREATE TRIGGER 文, 670
AFTER トリガー
CREATE TRIGGER 文, 670
ALGORITHM 句
CREATE ENCRYPTED FILE 文, 538
CREATE ENCRYPTED TABLE DATABASE 文, 535
ALL
SELECT 文内のキーワード, 897
ALL authority 句
GRANT 文, 784
ALLOCATE DESCRIPTOR 文
Embedded SQL 構文, 406
ALL permission 句
GRANT 文, 784
ALL PRIVILEGES permission 句
GRANT 文, 784
ALL PRIVILEGES 句
REVOKE 文, 887
ALL 句
REVOKE 文, 887
ALL 句
CREATE EVENT 文, 540
DESCRIBE 文, 701
DISCONNECT 文 [ESQL] [Interactive SQL] 文, 707
MEDIAN 関数, 277
SELECT 文, 898
ALL 探索条件
構文, 37
ALTER DATABASE UPGRADE 文
構文, 408
ALTER DATABASE 文
FORCE START 句, 412
SET PARTNER FAILOVER 句, 410
構文, 408
ALTER DATATYPE 文
構文, 416
ALTER DBSPACE 文
構文, 414
ALTER DOMAIN 文
構文, 416
ALTER EVENT 文
構文, 417
ALTER EXTERNAL ENVIRONMENT 文
構文, 420
ALTER FUNCTION 文
構文, 421
ALTER INDEX 文
構文, 422
ALTER LOGIN POLICY 文
構文, 424
ALTER MATERIALIZED VIEW 文
構文, 427
ALTER MIRROR SERVER 文
構文, 430
ALTER OPTION 句
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
ALTER SYNCHRONIZATION USER 文 [Mobile Link], 461
ALTER PARENT FROM 句
ALTER MIRROR SERVER 文, 431
ALTER permission 句
GRANT 文, 784
ALTER PROCEDURE 文
構文, 433
ALTER PUBLICATION 文
Mobile Link 構文, 435
SQL Remote 構文, 435
ALTER REMOTE MESSAGE TYPE 文
SQL Remote 構文, 436
ALTER SEQUENCE 文
構文, 437
ALTER SERVER 文
構文, 439
ALTER SERVICE 文 [HTTP Web サービス]

構文, 441
ALTER SERVICE 文 [SOAP Web サービス]
構文, 446
ALTER SPATIAL REFERENCE SYSTEM 文
構文, 453
ALTER STATISTICS 文
構文, 456
ALTER SYNCHRONIZATION PROFILE 文
Mobile Link 構文, 457
ALTER SYNCHRONIZATION SUBSCRIPTION 文
Mobile Link 構文, 458
ALTER SYNCHRONIZATION USER 文
Mobile Link 構文, 461
ALTER TABLE 文
構文, 462
ALTER TEXT CONFIGURATION 文
構文, 476
ALTER TEXT INDEX 文
構文, 480
ALTER TRANSACTION LOG 句
ALTER DATABASE 文, 409
ALTER TRIGGER 文
構文, 481
ALTER USER 文
構文, 482
パスワードの最大長, 483
パスワードの文字セット変換, 483
パスワード、文字セット変換, 483
ALTER VIEW 文
DISABLE 句, 485
ENABLE 句, 485
RECOMPILE 句, 485
構文, 484
ALTER 句
REVOKE 文, 887
ALTER 句
ALTER TABLE 文, 470
ALTER パーミッション
REVOKE 文, 888
ALTER パーミッション
GRANT 文, 782
AND
3 値的論理, 59
ビット処理演算子, 11
論理演算子の説明, 9
ANGULAR UNIT OF MEASURE 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
ANSI
REWRITE 関数を使用した書き換え, 332
ansi_nulls オプション
Microsoft SQL Server との互換性, 930
ansi_permissions オプション
Transact-SQL SET 文を使用して設定, 929
ansinull オプション
Transact-SQL SET 文を使用して設定, 929
ANY 探索条件
構文, 38
APPEND 句
OUTPUT 文, 849
UNLOAD 文, 967
ARGN 関数
構文, 153
ARRAY 句
EXECUTE 文, 743
FETCH 文, 751
PUT 文 [ESQL], 860
ASC | DESC 句
CREATE INDEX 文, 569
ASCII
関数と構文, 153
ASE COMPATIBLE 句
CREATE DATABASE 文, 522
ASIN 関数
構文, 154
AS 句
ALTER MIRROR SERVER 文, 430
ALTER SERVICE 文 [HTTP Web サービス], 441
ALTER SERVICE 文 [SOAP Web サービス], 446
ALTER VIEW 文, 484
CONNECT 文 [ESQL] [Interactive SQL], 517
CREATE DBSPACE 文, 528
CREATE DOMAIN 文, 533
CREATE FUNCTION 文, 563
CREATE MATERIALIZED VIEW 文, 576
CREATE MESSAGE 文, 578
CREATE MIRROR SERVER 文, 580
CREATE PROCEDURE 文 [T-SQL], 599
CREATE TRIGGER [Transact-SQL] 文, 676
CREATE VIEW 文, 681
DELETE 文, 697
START DATABASE 文, 935
UPDATE 文, 976

ATAN2 関数
構文, 156
ATAN 関数
構文, 155
ATN2 関数
構文, 156
ATOMIC 句
BEGIN 文, 497
ATTACH TRACING 文
構文, 487
診断トレーシング, 487
ATTENDED 句
BACKUP 文, 491
AT 句
ALTER EVENT 文, 418
CREATE EVENT 文, 544
CREATE EXISTING TABLE 文, 546
CREATE PROCEDURE 文, 605
CREATE TABLE 文, 653
GRANT 文, 783
authentication_string オプション
SET MIRROR OPTION 文, 909
AUTHORIZATION 句
ALTER SERVICE 文 [HTTP Web サービス], 441
ALTER SERVICE 文 [SOAP Web サービス], 446
CREATE SERVICE 文 [HTTP Web サービス], 443, 625
CREATE SERVICE 文 [SOAP Web サービス], 449, 631
auto_add_fan_out オプション
SET MIRROR OPTION 文, 909
auto_add_server オプション
SET MIRROR OPTION 文, 909
auto_commit オプション
Interactive SQL オプション, 916
auto_failover オプション
SET MIRROR OPTION 文, 909
AUTO COMPRESSED 句
LOAD TABLE 文, 819
AUTOINCREMENT
@@identity, 83
GET_IDENTITY 関数, 234
値のリセット, 1162
AUTOINCREMENT 句
CREATE TABLE 文, 651
AUTOINCREMENT デフォルト
CREATE TABLE 文, 465, 657
AutoMultiProgrammingLevelStatistics プロパティ
sa_server_option での設定, 1177
AutoMultiProgrammingLevel プロパティ
sa_server_option での設定, 1177
AUTO REFRESH 句
CREATE TEXT INDEX 文, 668
AUTOSTOP 句
START DATABASE 文, 935
AUTO TUNE WRITERS 句
BACKUP 文, 491
AUTO UPDATE 句
ALTER STATISTICS 文, 456
AUTO 句
BACKUP 文, 492
AvailForOptimization プロパティ
sa_materialized_view_info システムプロシージャー, 1119
AVG 関数
構文, 156

B

backup.syb
説明, 491
BACKUP authority 句
GRANT 文, 783
BACKUP 句
REVOKE 文, 887
BACKUP 権限
GRANT 文, 782
REVOKE 文, 888
BACKUP 文
構文, 489
BASE64_DECODE 関数
構文, 158
BASE64_ENCODE 関数
構文, 159
BEFORE キーワード、CONTAINS 探索条件
全文検索ではサポートされない, 54
BEFORE 句
CREATE TRIGGER 文, 670
BEGIN DECLARE 文
Embedded SQL 構文, 685
BEGIN SNAPSHOT 文
構文, 496
BEGIN TRANSACTION 文
Transact-SQL 構文, 500
BEGIN キーワード

TSQL 互換性, 499
BEGIN 文
Transact-SQL 構文, 499
構文, 497
BETWEEN ...AND 句
CREATE EVENT 文, 543
BETWEEN 句
CREATE SPATIAL REFERENCE SYSTEM 文, 635
WINDOW 句, 990
BETWEEN 探索条件
構文, 39
BIGINT データ型
構文, 94
日付と時刻の比較, 130
BINARY データ型
構文, 122
BIND VARIABLES FOR 句
DESCRIBE 文 [ESQL], 700
BIT_AND 関数
構文, 159
BIT_LENGTH 関数
構文, 160
BIT_OR 関数
構文, 161
BIT_SUBSTR 関数
構文, 162
BIT_XOR 関数
構文, 163
BIT データ型
構文, 95
BLANK PADDING 句
CREATE DATABASE 文, 522
BLOB
ALTER TABLE 文を使用して BLOB インデックスを設定, 469
ASE 生成の BCP ファイルのインポート, 821
BLOB 内のクエリ, 762
GET DATA 文, 771
INLINE 句、CREATE TABLE 文, 654
PREFIX 句、CREATE TABLE 文, 654
SET 文の例, 928
SET 文を使用して挿入, 927
xp_read_file システムプロシージャーを使用して挿入, 1239
エクスポート, 1250
テーブル作成時の BLOB インデックスの設定, 655
トランザクションログの考慮事項, 414
BLOB の挿入
xp_read_file システムプロシージャーを使用, 1239
BLOCK 句
FETCH 文, 750
OPEN 文, 846
BOM (バイトオーダーマーク)
CSCONVERT 関数の読み込み／書き込みオプション, 189
UTF-16 または UTF-8 データファイルからデータをロード, 826
BREAK 文
Transact-SQL 構文, 502
BYE 文
Interactive SQL 構文, 746
BY LOCK 句
PREPARE 文 [ESQL] 文, 857
BYTE_LENGTH 関数
構文, 164
BYTE_SUBSTR 関数
構文, 164
BYTE ORDER MARK 句
INPUT 文, 795
LOAD TABLE 文, 818
openxml システムプロシージャー, 1037
OUTPUT 文, 849
UNLOAD 文, 967
BY TIMESTAMP 句
PREPARE 文 [ESQL] 文, 857
BY VALUES 句
PREPARE 文 [ESQL] 文, 857
BY 句
INPUT 文, 795

C

CacheSizingStatistics プロパティ
sa_server_option での設定, 1177
CACHE 句
ALTER SEQUENCE 文, 438
CREATE SEQUENCE 文, 617
CALIBRATE [SERVER] 句
ALTER DATABASE 文, 409
CALIBRATE DBSPACE TEMPORARY 句
ALTER DATABASE 文, 409
CALIBRATE DBSPACE 句
ALTER DATABASE 文, 409

CALIBRATE GROUP READ 句
ALTER DATABASE 文, 409
CALIBRATE PARALLEL READ 句
ALTER DATABASE 文, 409
CALL 文
Transact-SQL, 744
構文, 503
CAPABILITY 句
ALTER SERVER 文, 439
CASCADE 句
CREATE TABLE 文, 651
CaseSensitivity プロパティ
DB_EXTENDED_PROPERTY 関数, 203
CASE クラス
CREATE DATABASE 文, 523
CASE 式
NULLIF 関数, 295
構文, 16
CASE 文
構文, 505
CAST 関数
構文, 165
データ型変換, 127
CatalogCollation プロパティ
DB_EXTENDED_PROPERTY 関数, 203
CATALOG ONLY 句
RESTORE DATABASE 文, 880
CEILING 関数
構文, 166
CERTIFICATE 句
CREATE FUNCTION 文 [Web クライアント], 559
CREATE PROCEDURE 文 [Web クライアント], 595
CHAR_LENGTH 関数
構文, 168
CHARINDEX 関数
構文, 169
CharsetConversion オプション
sa_set_http_option システムプロシージャー, 1186
CharSet プロパティ
DB_EXTENDED_PROPERTY 関数, 203
CHAR 関数
構文, 167
CHAR データ型
CHAR カラムに DESCRIBE を使用, 85
NCHAR データ型との比較, 129
構文, 85
バイト長のセマンティック, 85
日付と時刻の比較, 130
文字長のセマンティック, 85
CHECK CONSTRAINTS 句
LOAD TABLE 文, 818
CHECK EVERY 句
WAITFOR 文, 986
CHECK ON COMMIT 句
CREATE TABLE 文, 661
CHECKPOINT 文
構文, 509
CHECKSUM 句
ALTER DATABASE 文, 410
CREATE TRIGGER 文, 523
START DATABASE 文, 936
CHECK 句
ALTER TABLE 文, 467
CREATE DOMAIN 文, 533
CREATE MATERIALIZED VIEW 文, 576
CREATE TABLE 文, 661
探索条件, 34
CHECK 条件
CREATE TABLE 文, 659
child_creation オプション
SET MIRROR OPTION 文, 909
CLASS 句
ALTER SERVER 文, 439
CREATE SERVER 文, 619
REMOVE JAVA 文, 876
CLEAR 文
Interactive SQL 構文, 510
CLIENTPORT 句
CREATE FUNCTION 文 [Web クライアント], 559
CREATE PROCEDURE 文 [Web クライアント], 595
Clockwise フォーマット
CREATE SPATIAL REFERENCE SYSTEM 文, 640
close_on_endtrans オプション
Transact-SQL SET 文を使用して設定, 929
CLOSE 文
Embedded SQL 構文, 510
構文, 510
CLUSTERED 句
ALTER INDEX 文, 422
CREATE INDEX 文, 568

COALESCE 関数
構文, 170
COL_LENGTH 関数
構文, 150
COL_NAME 関数
構文, 150
COLLATION 句
CREATE DATABASE 文, 524
照合の適合化, 524
Collation プロパティ
DB_EXTENDED_PROPERTY 関数, 203
CollectStatistics プロパティ
sa_server_option での設定, 1178
COLUMN DELIMITED BY 句
UNLOAD 文, 967
column-name
一般的な SQL 構文要素, 403
COLUMN WIDTHS 句
INPUT 文, 796
OUTPUT 文, 849
COLUMN 句
DESCRIBE 文 [ESQL], 700
GET DATA 文, 771
COMMENTS INTRODUCED BY 句
LOAD TABLE 文, 818
COMMENT 文
構文, 512
COMMIT 文
2 フェーズの準備, 858
構文, 514
参照整合性, 1045
COMPARE 関数
構文, 171
照合の適合化, 171
COMPRESSED 句
ALTER TABLE 文, 468
CREATE TABLE 文, 654
LOAD TABLE 文, 819
UNLOAD 文, 967
COMPRESS 関数
構文, 172
COMPUTES 句
LOAD TABLE 文, 819
COMPUTE 句
ALTER TABLE 文, 469
CREATE TABLE 文, 661
condition
一般的な SQL 構文要素, 403
CONFIGURATION 句
CREATE TEXT INDEX 文, 668
CONFIGURE 文
Interactive SQL 構文, 516
CONFLICT 関数
構文, 173
Connection
HTTP_RESPONSE_HEADER 関数, 251
CONNECTION_EXTENDED_PROPERTY 関数
構文, 175
CONNECTION_PROPERTY 関数
構文, 178
connection_string オプション
ALTER MIRROR SERVER 文, 432
CREATE MIRROR SERVER 文, 581
CONNECTION CLOSE 句
ALTER SERVER 文, 439
connection-name
一般的な SQL 構文要素, 403
CONNECT TO 句
GRANT 文, 785
CONNECT TRACING 文
sa_diagnostic_tracing_level テーブルにデータを作成する, 1022
CONNECT 句
REVOKE 文, 887
CONNECT パーミッション
GRANT 文, 785
CONNECT 文
Embedded SQL 構文, 516
Interactive SQL 構文, 516
ConnsDisabledForDB プロパティ
sa_server_option での設定, 1178
ConnsDisabled プロパティ
sa_server_option での設定, 1178
ConsoleLogFile プロパティ
sa_server_option での設定, 1178
ConsoleLogMaxSize プロパティ
sa_server_option での設定, 1178
CONSOLIDATED 句
CREATE EVENT 文, 540
CONSOLIDATE パーミッション
REVOKE CONSOLIDATE 文, 884
付与, 776
CONSOLIDATE パーミッションの取り消し
SQL Remote CONSOLIDATE パーミッション, 884
CONSTRAINT 句

CREATE TABLE 文, 651
CONTAINS 句
FROM 句, 763
CONTAINS 探索条件
あいまい一致, 51
構文, 51
サポートされていない BEFORE キーワードの使用, 54
探索条件, 34
特殊文字を使用できる構文, 55
Content-Length
HTTP_RESPONSE_HEADER 関数, 251
Content-Type
HTTP_RESPONSE_HEADER 関数, 251
CONTINUE 句
WHENEVER 文 [ESQL], 987
CONTINUE 文
Transact-SQL 構文, 520
CONVERT USING 句
CREATE SPATIAL UNIT OF MEASURE 文, 642
CONVERT 関数
構文, 179
データ型変換, 127
Cookie
HTTP_HEADER 関数, 249
COORDINATE 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 638
COPY 句
BACKUP 文, 492
CORR 関数
構文, 181
COS 関数
構文, 182
COT 関数
構文, 183
COUNT_BIG 関数
構文, 185
COUNT_SET_BITS 関数
構文, 186
CounterClockwise フォーマット
CREATE SPATIAL REFERENCE SYSTEM 文, 640
COUNT 関数
構文, 183
COUNT 句
GET DESCRIPTOR 文 [ESQL], 773
SET DESCRIPTOR 文 [ESQL], 907
COVAR_POP 関数
構文, 187
COVAR_SAMP 関数
構文, 188
CREATE DATABASE 文
構文, 521
CREATE DATATYPE 文
構文, 533
CREATE DBSPACE 文
構文, 528
CREATE DECRYPTED DATABASE 文
構文, 530
CREATE DECRYPTED FILE 文
構文, 531
CREATEDIRS 句
CREATE SERVER 文, 620
CREATE DOMAIN 文
構文, 533
使用, 125
CREATE ENCRYPTED DATABASE 文
構文, 535
CREATE ENCRYPTED FILE 文
暗号化キーを変更する場合の使用例, 540
構文, 537
CREATE ENCRYPTED TABLE DATABASE 文
構文, 535
CREATE EVENT 句
CREATE EVENT 文, 541
CREATE EVENT 文
構文, 540
CREATE EXISTING TABLE 文
sp_remote_columns システムプロシージャー, 1219
sp_remote_tables システムプロシージャー, 1226
構文, 546
CREATE EXTERNLOGIN 文
構文, 548
CREATE FUNCTION 文
Transact-SQL の例, 567
Web サービス機能を作成するための構文, 555
構文, 563
ネイティブ関数へのインターフェイスを作成するための構文, 550
CREATE INDEX 文
構文, 567

テーブルの使用, 570
CREATE LOCAL TEMPORARY TABLE 文
構文, 571
CREATE LOGIN POLICY 文
構文, 573
CREATE MATERIALIZED VIEW 文
構文, 576
CREATE MESSAGE 文
Transact-SQL 構文, 578
CREATE MIRROR SERVER 文
構文, 579
CREATE ON 句
REVOKE 文, 887
CREATE ON 句
GRANT 文, 785
CREATE PROCEDURE 文
Transact-SQL 構文, 599
Web サービスプロシージャーを作成するための構文, 591
構文, 602
ネイティブプロシージャーへのインターフェイスを作成するための構文, 583
CREATE PUBLICATION 文
Mobile Link 構文, 610
SQL Remote 構文, 610
CREATE REMOTE MESSAGE TYPE 文
SQL Remote 構文, 613
CREATE SCHEMA 文
構文, 615
CREATE SEQUENCE 文
構文, 616
CREATE SERVER 文
構文, 618
CREATE SERVICE 文 [HTTP Web サービス]
構文, 622
CREATE SERVICE 文 [SOAP Web サービス]
構文, 628
CREATE SPATIAL REFERENCE SYSTEM 文
構文, 635
CREATE SPATIAL UNIT OF MEASURE 文
構文, 642
CREATE STATISTICS 文
構文, 643
CREATE SUBSCRIPTION 文
SQL Remote 構文, 645
CREATE SYNCHRONIZATION PROFILE 文
Mobile Link 構文, 646
CREATE SYNCHRONIZATION SUBSCRIPTION 文
Mobile Link 構文, 647
CREATE SYNCHRONIZATION USER 文
Mobile Link 構文, 650
CREATE TABLE 句
INPUT 文, 796
OUTPUT 文, 849
CREATE TABLE 文
Transact-SQL, 663
構文, 651
リモートテーブル, 653
CREATE TEMPORARY PROCEDURE 文
構文, 602
CREATE TEXT CONFIGURATION 文
構文, 666
CREATE TEXT INDEX 文
構文, 667
CREATE TRIGGER 文
Transact-SQL 構文, 676
構文, 670
トリガーオペレーション条件, 673
CREATE USER 文
構文, 677
パスワードの最大長, 678
パスワードの文字セット変換, 678
パスワード、文字セット変換, 678
CREATE VARIABLE 文
構文, 679
CREATE VIEW 文
構文, 681
CROSS APPLY 句
FROM 句, 763
CROSS JOIN 句
FROM 句 SQL 構文, 758
CSCONVERT 関数
構文, 189
CUBE 演算
GROUP BY 句, 789
WITH CUBE 句, 789
CUME_DIST 関数
構文, 191
CURRENT_TIMESTAMP 特別値
構文, 64
CURRENT_USER 特別値
構文, 65
CURRENT DATABASE 句
CREATE TABLE 文, 651

CURRENT DATABASE 特別値
構文, 62
CURRENT DATE 関数
TODAY 関数, 372
CURRENT DATE 句
CREATE TABLE 文, 651
CURRENT DATE 特別値
構文, 62
CurrentMultiProgrammingLevel プロパティ
sa_server_option での設定, 1178
CURRENT PUBLISHER 句
CREATE TABLE 文, 651
CURRENT PUBLISHER データ型
設定, 777
CURRENT PUBLISHER 特別値
構文, 62
CURRENT REMOTE USER 句
CREATE TABLE 文, 651
CURRENT REMOTE USER 特別値
構文, 63
CURRENT TIMESTAMP 句
CREATE TABLE 文, 651
CURRENT TIMESTAMP デフォルト
CREATE TABLE 文, 465, 656
CURRENT TIMESTAMP 特別値
構文, 64
CURRENT TIME 句
CREATE TABLE 文, 651
CURRENT TIME 特別値
構文, 63
CURRENT USER 句
CREATE TABLE 文, 651
CURRENT USER 特別値
構文, 65
CURRENT UTC TIMESTAMP 句
CREATE TABLE 文, 651
CURRENT UTC TIMESTAMP デフォルト
CREATE TABLE 文, 465, 656
CURRENT UTC TIMESTAMP 特別値
構文, 66
CURRENT 句
DISCONNECT 文 [ESQL] [Interactive SQL] 文, 707
CYCLE 句
ALTER SEQUENCE 文, 438
CREATE SEQUENCE 文, 617

D

DatabaseCleaner プロパティ
sa_server_option での設定, 1178
DATABASE SIZE 句
CREATE DATABASE 文, 524
DATABASE 句
CONNECT 文, 516
DataLastModified プロパティ
sa_materialized_view_info システムプロシージャー, 1119
DATALENGTH 関数
構文, 192
data-type
SQL 構文のカラム要素, 403
DATATYPEP 句
ALTER SERVICE 文 [SOAP Web サービス], 446
CREATE SERVICE 文 [SOAP Web サービス], 448, 630
DATA 句
GET DESCRIPTOR 文 [ESQL], 773
SET DESCRIPTOR 文 [ESQL], 907
Date
HTTP_RESPONSE_HEADER 関数, 251
DATEADD 関数
構文, 194
DATEDIFF 関数
構文, 196
datefirst オプション
SET 文の構文, 929
DATEFORMAT 関数
構文, 197
DATENAME 関数
構文, 198
DATEPART 関数
構文, 199
datetime
変換関数, 142
DATETIMEOFFSET データ型
構文, 114
日付と時刻をデータベースに送信する, 106
DATETIME 関数
構文, 200
DATETIME データ型
構文, 113
日付と時刻をデータベースに送信する, 106
DATE 関数

構文, 193
DATE データ型
ISO 8601 週日, 107
ISO 8601 年内の日の番号, 107
ISO 8601 暦日, 107
あいまいな日付の解決, 108
構文, 112
日付と時刻の比較, 130
日付と時刻をデータベースに送信する, 106
DAYNAME 関数
構文, 201
DAYS 関数
構文, 202
DAY 関数
構文, 201
db_charset
CSCONVERT 関数, 189
DB_EXTENDED_PROPERTY 関数
構文, 203
DB_ID 関数
構文, 207
DB_NAME 関数
構文, 208
DB_PROPERTY 関数
構文, 208
db_register_a_callback 関数
MESSAGE TO CLIENT と使用, 844
DB2
sa_migrate システムプロシージャーを使用して SQL Anywhere にマイグレート, 1128
DBA
GRANT 文での権限の付与, 783
DBA authority 句
GRANT 文, 783
DBA PASSWORD 句
CREATE DATABASE 文, 524
DBA USER 句
CREATE DATABASE 文, 524
DBA 句
REVOKE 文, 887
DBA 権限
REVOKE 文, 888
DBA 権限
GRANT 文, 782
DBFILE ONLY 句
BACKUP 文, 490
DBFreePercent イベント条件
説明, 222
DBFreeSpace イベント条件
説明, 222
dbmlsync ユーティリティ
ALTER EXTERNAL ENVIRONMENT 文を使用したロケーションの指定, 420
dbname FORCE START 句
ALTER DATABASE 文, 410
dbo ユーザー
RowGenerator システムテーブル, 1027
Transact-SQL 互換ビュー, 1365
DBSize イベント条件
説明, 222
DB 領域
ALTER DBSPACE 文を使用した変更, 414
COMMENT 文を使用してコメントを追加, 512
CREATE DBSPACE 文を使用して作成, 528
CREATE DECRYPTED DATABASE 文を使用した復号化, 530
CREATE DECRYPTED FILE 文を使用した復号化, 531
CREATE ENCRYPTED DATABASE 文を使用した暗号化, 535
CREATE ENCRYPTED FILE 文を使用した暗号化, 537
DROP DBSPACE 文を使用して削除, 711
SYSFILE システムビュー, 1358
使用可能な領域の決定, 1080
使用パーミッションの付与, 783
ページの追加, 414
deadlock_logging プロパティ
sa_server_option での設定, 1178
DeadlockLogging プロパティ
sa_server_option での設定, 1178
DEALLOCATE DESCRIPTOR 文
Embedded SQL 構文, 684
DEALLOCATE 文
構文, 685
DebuggingInformation プロパティ
sa_server_option での設定, 1178
DEBUG ONLY 句
MESSAGE 文, 842
DECIMAL データ型
構文, 96
DECLARE CURSOR 文
Embedded SQL 構文, 686
Transact-SQL 構文, 690
構文, 686
DECLARE EXCEPTION

BEGIN 文とともに使用, 497
DECLARE LOCAL TEMPORARY TABLE 文
構文, 691
DECLARE 文
BEGIN 文とともに使用, 497
BEGIN 文とともに使用 [TSQL], 499
構文, 693
DECOMPRESS 関数
構文, 209
DECRYPT 関数
構文, 211
DEFAULT
CALL 文で指定, 503
DEFAULT LAST USER
SQL Remote でカラムをレプリケートしない, 804
DEFAULTS 句
LOAD TABLE 文, 819
DEFAULT TIMESTAMP カラム
TIMESTAMP 特別値, 73
DEFAULT VALUES 句
INSERT 文, 802
DEFAULT 句
ALTER TABLE 文, 464
CREATE DOMAIN 文, 533
CREATE SERVICE 文 [HTTP Web サービス], 622
CREATE SERVICE 文 [SOAP Web サービス], 628
CREATE TABLE 文, 656
特別値, 465, 656
DEFAULT ログインポリシー
説明, 573
DEFINITION 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 636
DEGREES 関数
構文, 212
DELAY 句
WAITFOR 文, 985
DELETE ALL OPTION 句
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
ALTER SYNCHRONIZATION USER 文 [Mobile Link], 461
DELETE OPTION 句
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
ALTER SYNCHRONIZATION USER 文 [Mobile Link], 461
DELETE permission 句
GRANT 文, 784
DELETE TABLE 句
ALTER PUBLICATION 文, 435
DELETE TYPE 句
ALTER EVENT 文, 418
DELETE (位置付け) 文
Embedded SQL 構文, 695
DELETE 句
REVOKE 文, 887
DELETE 句
CREATE LOCAL TEMPORARY TABLE 文, 571
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
CREATE TRIGGER 文, 671
DECLARE LOCAL TEMPORARY TABLE 文, 691
MERGE 文, 837
DELETE パーミッション
REVOKE 文, 888
DELETE パーミッション
GRANT 文, 782
DELETE 文
(位置付け) 文の構文, 695
構文, 697
データベースオプションの設定, 699
DELIMITED BY 句
LOAD TABLE 文, 819
OUTPUT 文, 849
UNLOAD 文, 967
DELIMITED 句
INPUT 文, 796
DENSE_RANK 関数
構文, 213
DESCRIBE CONNECTION 文
Interactive SQL 構文, 704
DESCRIBE 句
PREPARE 文, 856
DESCRIBE 文
DT_DESCRIBE_INPUT, 701
DT_HIDDEN_COLUMN, 686
DT_KEY_COLUMN, 686
DT_PROCEDURE_IN, 701
DT_PROCEDURE_OUT, 701

Embedded SQL 構文, 700
Interactive SQL 構文, 704
長いカラム名, 702
DESCRIPTOR 句
EXECUTE 文, 742
DETACH TRACING 文
構文, 706
診断トレーシング, 706
DIFFERENCE 関数
構文, 215
DIRECTORY 句
BACKUP 文, 489
START DATABASE 文, 936
DISABLE USE IN OPTIMIZATION 句
ALTER MATERIALIZED VIEW 文, 427
DISABLE VIEW DEPENDENCIES 句
ALTER TABLE 文, 473
DISABLE 句
ALTER EVENT 文, 417
ALTER MATERIALIZED VIEW 文, 427
ALTER SERVICE 文 [HTTP Web サービス], 441
ALTER SERVICE 文 [SOAP Web サービス], 446
ALTER VIEW 文, 485
CREATE SERVICE 文 [HTTP Web サービス], 443, 625
CREATE SERVICE 文 [SOAP Web サービス], 450, 632
DISCONNECT 文
Embedded SQL 構文, 707
Interactive SQL 構文, 707
DISH
CREATE SERVICE 文 [SOAP Web サービス], 447, 629
DISH サービス
フォワードスラッシュは名前に使用できない, 447, 629
DISTINCT キーワード
説明, 897
DISTINCT 句
MEDIAN 関数, 277
NULL, 69
SELECT 文, 898
DML
選択に使用, 763
DML から選択
FROM 句, 763
DOMAIN | DATATYPE 句
CREATE DOMAIN 文, 533
DOUBLE データ型
NUMERIC への変換, 135
構文, 97
日付と時刻の比較, 130
DOW 関数
構文, 215
DO 句
FOR 文, 753
DRIVER 句
INPUT 文, 794
DriveType プロパティ
DB_EXTENDED_PROPERTY 関数, 203
DropBadStatistics プロパティ
sa_server_option での設定, 1178
DROP CONNECTION 文
構文, 709
DROP DATABASE 文
構文, 709
DROP DATATYPE 文
構文, 710
DROP DBSPACE 文
構文, 711
DROP DOMAIN 文
構文, 712
DROP EVENT 文
構文, 713
DROP EXTERNLOGIN 文
構文, 713
DROP FUNCTION 文
構文, 714
DROP INDEX 文
構文, 715
DROP LOGIN POLICY 文
構文, 716
DROP MATERIALIZED VIEW 文
構文, 717
DROP MESSAGE 文
構文, 718
DROP MIRROR SERVER 文
構文, 719
DROP PCTFREE 句
ALTER MATERIALIZED VIEW 文, 428
DROP PREFILTER 句
ALTER TEXT CONFIGURATION 文, 478
DROP PROCEDURE 文
構文, 720
DROP PUBLICATION 文

Mobile Link 構文, 721
SQL Remote 構文, 721
DROP REMOTE MESSAGE TYPE 文
SQL Remote 構文, 722
DROP SEQUENCE 文
構文, 722
DROP SERVER 文
構文, 723
DROP SERVICE 文
構文, 724
DROP SPATIAL REFERENCE SYSTEM 文
構文, 724
DROP SPATIAL UNIT OF MEASURE 文
構文, 725
DROP STATEMENT 文
Embedded SQL 構文, 726
DROP STATISTICS 文
構文, 727
DROP STOPLIST 句
ALTER TEXT CONFIGURATION 文, 477
DROP SUBSCRIPTION 文
SQL Remote 構文, 728
DROP SYNCHRONIZATION PROFILE 文
Mobile Link 構文, 729
DROP SYNCHRONIZATION SUBSCRIPTION 文
構文, 729
DROP SYNCHRONIZATION USER 文
Mobile Link 構文, 731
DROP TABLE 句
ALTER PUBLICATION 文, 435
DROP TABLE 文
構文, 731
DROP TEXT CONFIGURATION 文
構文, 733
DROP TEXT INDEX 文
構文, 733, 734
DROP TRIGGER 文
構文, 734
DropUnusedStatistics プロパティ
sa_server_option での設定, 1178
DROP USER 文
構文, 735
DROP VARIABLE 文
構文, 736
DROP VIEW 文
構文, 737
DROP 句
ALTER TABLE 文, 472
DROP EXTERNLOGIN 文, 714
DSN 句
INPUT 文, 794
DT_DESCRIBE_INPUT
DESCRIBE 文, 701
DT_HIDDEN_COLUMN
DESCRIBE 文, 686
DT_KEY_COLUMN
DESCRIBE 文, 686
DT_PROCEDURE_IN
DESCRIBE 文, 701
DT_PROCEDURE_OUT
DESCRIBE 文, 701
DUMMY
システムテーブル, 993
ローコンストラクターアルゴリズム, 993
DYNAMIC RESULT SETS 句
CREATE PROCEDURE 文 [外部呼び出し], 586
DYNAMIC SCROLL カーソル
宣言, 686
DYNAMIC SCROLL 句
DECLARE CURSOR 文, 687
FOR 文, 753

E

ELLIPSOID 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 639
ELSE
CASE 式, 16
IF 式, 15
ELSEIF 句
IF 文, 791
ELSE 句
IF 文, 791
IF 文 [T-SQL], 793
Embedded SQL
ALLOCATE DESCRIPTOR 構文, 406
BEGIN DECLARE SQL 構文, 685
CLOSE SQL 構文, 510
CONNECT SQL 構文, 516
DEALLOCATE DESCRIPTOR SQL 構文, 684
DECLARE CURSOR SQL 構文, 686
DELETE (位置付け) SQL 構文, 695
DESCRIBE SQL 構文, 700

DISCONNECT SQL 構文, 707
DROP STATEMENT SQL 構文, 726
END DECLARE SQL 構文, 685
EXECUTE IMMEDIATE SQL 構文, 740
EXECUTE SQL 構文, 742
EXPLAIN SQL 構文, 747
FETCH SQL 構文, 749
GET DATA SQL 構文, 771
GET DESCRIPTOR SQL 構文, 773
GET OPTION SQL 構文, 774
INCLUDE SQL 構文, 794
OPEN SQL 構文, 844
PREPARE SQL 構文, 856
PUT SQL 構文, 860
SET CONNECTION SQL 構文, 906
SET DESCRIPTOR SQL 構文, 907
SET SQLCA SQL 構文, 926
UPDATE (位置付け) 文, 971
WHENEVER SQL 構文, 987
ENABLE | DISABLE 句
CREATE EVENT 文, 544
ENABLE USE IN OPTIMIZATION 句
ALTER MATERIALIZED VIEW 文, 427
ENABLE 句
ALTER MATERIALIZED VIEW 文, 427
ALTER SERVICE 文 [HTTP Web サービス], 441
ALTER SERVICE 文 [SOAP Web サービス], 446
ALTER VIEW 文, 485
CREATE SERVICE 文 [HTTP Web サービス], 443, 625
CREATE SERVICE 文 [SOAP Web サービス], 450, 632
encoding
INPUT 文, 796
OUTPUT 文, 849
READ 文, 864
ENCODING 句
CREATE DATABASE 文, 524
INPUT 文, 796
LOAD TABLE 文, 819
OUTPUT 文, 849
READ 文 [Interactive SQL], 863
UNLOAD 文, 967
ENCRYPTED DATABASE 句
CREATE ENCRYPTED DATABASE 文, 535
ENCRYPTED TABLE DATABASE 句
CREATE ENCRYPTED TABLE DATABASE 文, 535
ENCRYPTED TABLE 句
CREATE EXISTING TABLE 文, 525
ENCRYPTED 句
ALTER MATERIALIZED VIEW 文, 428
CREATE TABLE 文, 653
CREATE TRIGGER 文, 525
LOAD TABLE 文, 820
UNLOAD 文, 967
ENCRYPT 関数
構文, 216
END
CASE 式, 16
END CASE
CASE 式, 16
END DECLARE 文
Embedded SQL 構文, 685
END FOR 句
FOR 文, 753
ENDIF
IF 式, 15
END IF
IF 式, 15
ENDIF 句
IF 文, 791
END IF 句
IF 文, 791
END LOOP 句
LOOP 文, 832
END キーワード
TSQL 互換性, 499
END 文
BEGIN 文とともに使用, 497
BEGIN 文とともに使用 [TSQL], 499
ENVIRONMENT 句
INSTALL EXTERNAL OBJECT 文, 807
ERRORMSG 関数
構文, 218
ErrorNumber イベント条件
説明, 222
ESCAPE CHARACTER 句
INPUT 文, 797
LOAD TABLE 文, 820
OUTPUT 文, 849
UNLOAD 文, 968
ESCAPES 句
INPUT 文, 797

LOAD TABLE 文, 820
OUTPUT 文, 850
UNLOAD 文, 967
ESCAPE 句
LIKE 探索条件, 42
ESQL
文インジケーター, 406
ESTIMATE_SOURCE 関数
構文, 220
ESTIMATE 関数
構文, 219
EvenOdd フォーマット
CREATE SPATIAL REFERENCE SYSTEM 文, 640
EVENT_CONDITION_NAME 関数
構文, 223
EVENT_CONDITION 関数
構文, 222
EVENT_PARAMETER 関数
構文, 223
EVERY 句
CREATE EVENT 文, 543
EXCEPTION 句
BEGIN 文, 497
EXCEPT 句
データベースオプションの設定, 738
EXCEPT 文
構文, 738
EXECUTE IMMEDIATE 文
構文, 740
EXECUTE LOGIN パーミッション
GRANT 文, 782
EXECUTE 文
Embedded SQL 構文, 742
Transact-SQL 構文, 744
Transact-SQL 構文、説明, 744
EXEC 文
Transact-SQL 構文, 744
EXISTS 探索条件
構文, 57
探索条件, 34
EXIT 文
Interactive SQL 構文, 746
EXPERIENCE_ESTIMATE 関数
構文, 227
Expires
HTTP_RESPONSE_HEADER 関数, 251
EXPLAIN 文
Embedded SQL 構文, 747
EXPLANATION 関数
構文, 228
expression
一般的な SQL 構文要素, 403
EXPRTYPE 関数
構文, 229
EXP 関数
構文, 226
EXTERNAL NAME 句
ALTER TEXT CONFIGURATION 文, 476
CREATE FUNCTION 文 [外部プロシージャー], 551
CREATE PROCEDURE 文 [外部プロシージャー], 587

F

FALSE 条件
3 値的論理, 59
IS FALSE 探索条件, 58
FASTFIRSTROW テーブルヒント
FROM 句, 766
FETCH 文
Embedded SQL 構文, 749
構文, 749
filename
一般的な SQL 構文要素, 403
FileSize プロパティ
DB_EXTENDED_PROPERTY 関数, 203
FILE 句
DROP REMOTE MESSAGE TYPE 文, 722
GRANT CONSOLIDATE 文 [SQL Remote], 776
File プロパティ
DB_EXTENDED_PROPERTY 関数, 203
FILE メッセージタイプ
SQL Remote ALTER REMOTE MESSAGE TYPE 文, 436
SQL Remote CREATE REMOTE MESSAGE TYPE 文, 613
filler() カラム名
FROM 句, 762
LOAD TABLE 文, 814
FIRST_VALUE 関数
構文, 230
FIRST 句
DELETE 文, 698
FETCH 文, 750

SELECT 文, 898
UPDATE 文, 978
FLOAT データ型
構文, 98
日付と時刻の比較, 130
FLOOR 関数
構文, 232
FOLLOWING 句
WINDOW 句, 990
FORCE BUILD 句
REFRESH MATERIALIZED VIEW 文, 868
REFRESH TEXT INDEX 文, 871
FORCE INCREMENTAL 句
REFRESH TEXT INDEX 文, 871
FORCE INDEX
インデックスヒント, 766
FORCE NO OPTIMIZATION 句
DELETE 文, 699
INSERT 文, 804
SELECT 文, 902
UPDATE 文, 979
FORCE OPTIMIZATION 句
DELETE 文, 699
EXCEPT 文, 738
INSERT 文, 804
INTERSECT 文, 811
MERGE 文, 839
SELECT 文, 902
UNION 文, 964
UPDATE 文, 979
FORCE PASSWORD CHANGE 句
ALTER USER 文, 483
CREATE USER 文, 678
FORCE START 句
ALTER DATABASE 文, 412
FORCE 句
REFRESH TEXT INDEX 文, 871
FOR DELETE 句
CREATE TRIGGER [Transact-SQL] 文, 676
FOR DOWNLOAD ONLY 句
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
FOR EACH 句
CREATE TRIGGER 文, 672
FOREIGN KEY 句
ALTER INDEX 文, 422
CREATE TABLE 文, 651
REORGANIZE TABLE 文, 877
VALIDATE 文, 983
FOR INSERT 句
CREATE TRIGGER [Transact-SQL] 文, 676
FORMAT BCP 句
LOAD TABLE 文, 821
FORMAT FIXED 句
INPUT 文, 797
FORMAT SHAPEFILE 句
INPUT 文, 798
LOAD TABLE 文, 821
FORMAT TEXT 句
INPUT 文, 797
LOAD TABLE 文, 821
FORMAT XML 句
LOAD TABLE 文, 821
FORMAT 句
ALTER SERVICE 文 [SOAP Web サービス], 446
CREATE SERVICE 文 [SOAP Web サービス], 448, 630
INPUT 文, 797
LOAD TABLE 文, 821
OUTPUT 文, 850
UNLOAD 文, 968
FOR OLAP WORKLOAD オプション
ALTER TABLE 文, 462
FOR OLAP WORKLOAD 句
CREATE INDEX 文, 569
CREATE TABLE 文, 662
FOR READ ONLY 句
DECLARE CURSOR 文 [T-SQL], 690
FOR 文, 754
PREPARE 文 [ESQL] 文, 857
SELECT 文, 900
START DATABASE 文, 935
FOR TABLES 句
START SYNCHRONIZATION SCHEMA CHANGE 文, 944
FOR UPDATE BY LOCK 句
SELECT 文, 901
FOR UPDATE BY TIMESTAMP 句
SELECT 文, 901
FOR UPDATE BY VALUES 句
SELECT 文, 901
FOR UPDATE BY 句
SELECT 文, 901
FOR UPDATE OF 句
SELECT 文, 902

FOR UPDATE 句
CREATE TRIGGER [Transact-SQL] 文, 676
DECLARE CURSOR 文 [T-SQL], 690
FETCH 文, 751
FOR 文, 754
PREPARE 文 [ESQL] 文, 857
SELECT 文, 900
FOR UPLOAD 句
ALTER PUBLICATION 文, 435
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
FORWARD TO 文
構文, 757
FOR XML 句
EXCEPT 文, 738
INTERSECT 文, 810
SELECT 文, 902
FOR 句
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
CREATE SUBSCRIPTION 文 [SQL Remote], 645
CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 648
CREATE TRIGGER [Transact-SQL] 文, 676
DECLARE CURSOR 文 [T-SQL], 690
DESCRIBE 文 [ESQL], 700
DROP SUBSCRIPTION 文 [SQL Remote], 728
DROP SYNCHRONIZATION SUBSCRIPTION 文, 730
MESSAGE 文, 842
SELECT 文, 900
START SUBSCRIPTION 文 [SQL Remote], 941
STOP SUBSCRIPTION 文 [SQL Remote], 950
SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote], 958
FOR 文
構文, 753
FREE PAGE ELIMINATION 句
BACKUP 文, 493
FreePages プロパティ
DB_EXTENDED_PROPERTY 関数, 203
FROM FILE 句
INSTALL EXTERNAL OBJECT 文, 807
INSTALL JAVA 文, 809
FROM SERVER 句
ALTER MIRROR SERVER 文, 431
CREATE MIRROR SERVER 文, 581
FROM VALUE 句
INSTALL EXTERNAL OBJECT 文, 807
FROM 句
REVOKE 文, 887
FROM 句
CREATE DECRYPTED DATABASE 文, 530
CREATE DECRYPTED FILE 文, 531
CREATE ENCRYPTED DATABASE 文, 535
CREATE ENCRYPTED FILE 文, 538
CREATE ENCRYPTED TABLE DATABASE 文, 535
CREATE TEXT CONFIGURATION 文, 666
DELETE (位置付け) 文 [ESQL] [SP] 文, 695
DELETE 文, 698
DML からの選択, 763
GRANT 文, 785
INSTALL EXTERNAL OBJECT 文, 807
INSTALL JAVA 文, 810
LOAD TABLE 文, 816
PUT 文 [ESQL], 860
RESTORE DATABASE 文, 880
REVOKE REMOTE DBA 文 [SQL Remote], 886
SELECT 文, 900
UPDATE (位置付け) 文, 971
UPDATE 文, 978
構文, 758
ストアドプロシージャーからの選択, 761
FTP 句
GRANT CONSOLIDATE 文 [SQL Remote], 776
FTP メッセージタイプ
SQL Remote ALTER REMOTE MESSAGE TYPE 文, 436
SQL Remote CREATE REMOTE MESSAGE TYPE 文, 613

G

GET_BIT 関数
構文, 233
GET_IDENTITY 関数
構文, 234
GET DATA 文
Embedded SQL 構文, 771
GETDATE 関数
現在のシステム日付のクエリ, 236
構文, 235
GET DESCRIPTOR 文
Embedded SQL 構文, 773
GET OPTION 文

Embedded SQL 構文, 774
global_database_id オプション
CREATE TABLE 文, 466, 657
GLOBAL AUTOINCREMENT
CREATE EVENT 文を使用してイベントを作成, 540
GLOBAL AUTOINCREMENT 句
CREATE TABLE 文, 651
GLOBAL AUTOINCREMENT デフォルト
CREATE TABLE 文, 466, 657
GOTO 句
WHENEVER 文 [ESQL], 987
GOTO 文
Transact-SQL 構文, 775
GRANT CONNECT 文
GRANT 文, 783
GRANT CONSOLIDATE 文
SQL Remote 構文, 776
GRANT CREATE ON 文
構文, 782
GRANT PUBLISH 文
SQL Remote 構文, 777
GRANT REMOTE DBA 文
Mobile Link 構文, 778
SQL Remote 構文, 778
GRANT REMOTE 文
SQL Remote 構文, 780
GRANT USAGE ON SEQUENCE 句
GRANT 文, 785
GRANT 文
構文, 782
パスワードの最大長, 785
パスワードの文字セット変換, 785
パスワード、文字セット変換, 785
パーミッションの確認, 1257
GRAPHICAL_PLAN 関数
構文, 236
GREATER 関数
構文, 238
GROUP BY 句
CUBE 文, 789
GROUPING SETS 演算, 788
ROLLUP 演算, 789
SELECT 文, 900
構文, 788
GROUPING SETS 演算
GROUP BY 句, 788
GROUPING 関数
構文, 238
GROUP 句
REVOKE 文, 887
GROUP 句
ALTER SERVICE 文 [SOAP Web サービス], 446
CREATE SERVICE 文 [SOAP Web サービス], 448, 629
GRANT 文, 784
GUID
NEWID 関数の SQL 構文, 286
STRTOUUID 関数の SQL 構文, 359
UNIQUEIDENTIFIER データ型, 123
UUIDTOSTR 関数の SQL 構文, 383
gzip ユーティリティ
COMPRESS 関数, 173
DECOMPRESS 関数, 210

H

HANDLER 句
CREATE EVENT 文, 544
HASH 関数
構文, 239
HAVING 句
SELECT 文, 900
探索条件, 34
HEADER 句
CREATE FUNCTION 文 [Web クライアント], 557
CREATE PROCEDURE 文 [Web クライアント], 595
HELP 文
Interactive SQL 構文, 791
HEXADECIMAL 句
LOAD TABLE 文, 823
OUTPUT 文, 850
UNLOAD 文, 968
HEXTOINT 関数
構文, 241
HISTORY 句
ATTACH TRACING 文, 487
BACKUP 文, 491
RESTORE DATABASE 文, 880
HOLDLOCK 句
READTEXT 文 [T-SQL], 866
HOLDLOCK テーブルヒント
FROM 句, 764

Host
HTTP_HEADER 関数, 249
hostvar
一般的な SQL 構文要素, 403
HOURS 関数
構文, 242
HOUR 関数
構文, 242
HTML
CREATE SERVICE 文 [HTTP Web サービス], 441, 623
HTML_DECODE 関数
構文, 244
HTML_ENCODE 関数
構文, 245
HTTP
SOAP ヘッダーの設定オプション, 1190
オプションの設定, 1186
ヘッダーの設定, 1185
HTTP_BODY 関数
構文, 246
HTTP_DECODE 関数
構文, 247
HTTP_ENCODE 関数
構文, 248
HTTP_HEADER 関数
構文, 249
HTTP_RESPONSE_HEADER 関数
構文, 251
HTTP_VARIABLE 関数
構文, 253
HttpMethod
HTTP_HEADER 関数, 250
HttpQueryString
HTTP_HEADER 関数, 250
HttpStatus ヘッダー
HTTP_RESPONSE_HEADER 関数, 252
sa_set_http_header_info システムプロシージャー, 1185
HttpURI
HTTP_HEADER 関数, 250
HttpVersion
HTTP_HEADER 関数, 250
HTTP 応答ヘッダー
@HttpStatus, 252
Connection, 251
Content-Length, 251
Content-Type, 251
Date, 251
Expires, 251
Location, 251
sa_set_http_header_info システムプロシージャー, 1185
Server, 251
Transfer-Encoding, 251
User-Agent, 251
WWW-Authenticate, 251
HTTP 関数
アルファベット順リスト, 147
HTTP サービス
CREATE SERVICE 文 [HTTP Web サービス] を使用して作成, 622
HTTP システムプロシージャー
アルファベット順リスト, 1030
HTTP ヘッダー
HTTP_HEADER 関数, 249
HTTP_RESPONSE_HEADER 関数, 251
HTTP 要求ヘッダー
@HttpMethod, 250
@HttpQueryString, 250
@HttpURI, 250
@HttpVersion, 250
Accept-Encoding, 249
Cookie, 249
Host, 249
Referer, 249
User-Agent, 249
名前と値を返す, 1097
HTTP を介した SOAP サービス
CREATE SERVICE 文 [SOAP Web サービス], 628

I

I/O
I/O コストモデルの再調整, 411
IDENTIFIED BY password 句
ALTER USER 文, 483
CONNECT 文, 516
IDENTIFIED BY 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE EXTERNLOGIN 文, 549
CREATE SPATIAL REFERENCE SYSTEM 文, 636
CREATE USER 文, 677

GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote], 779
IDENTITY カラム
@@identity, 83
IDENTITY 関数
構文, 254
IDENTITY 句
ALTER TABLE 文, 469
CREATE TABLE 文, 658
IdleTimeout プロパティ
sa_server_option での設定, 1179
IdleTime イベント条件
説明, 222
IF EXISTS 句
DROP EVENT 文, 713
DROP EXTERNLOGIN 文, 714
DROP INDEX 文, 715
DROP MATERIALIZED VIEW 文, 717
DROP PROCEDURE 文, 720
DROP PUBLICATION 文 [Mobile Link] [SQL Remote], 721
DROP SPATIAL REFERENCE SYSTEM 文, 724
DROP SPATIAL UNIT OF MEASURE 文, 725
DROP SYNCHRONIZATION PROFILE 文, 729
DROP TABLE 文, 731
DROP TRIGGER 文, 734
DROP VARIABLE 文, 736
DROP VIEW 文, 737
IF NOT EXISTS 句
CREATE INDEX 文, 568
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
CREATE TABLE 文, 654
CREATE TEXT INDEX 文, 668
IFNULL 関数
構文, 255
IF UPDATE 句
CREATE TRIGGER [Transact-SQL] 文, 676
Transact-SQL のトリガー, 676
トリガー内, 670
IF 式
構文, 15
探索条件, 34
IF 文
Transact-SQL 構文, 793
構文, 791
IMAGE データ型
構文, 123
IMMEDIATE REFRESH 句
ALTER MATERIALIZED VIEW 文, 428, 576
CREATE TEXT INDEX 文, 668
IN | ON 句
CREATE INDEX 文, 569
INCLUDE 文
Embedded SQL 構文, 794
INCREMENT BY 句
ALTER SEQUENCE 文, 438
CREATE SEQUENCE 文, 617
INDEX_COL 関数
構文, 150
INDEX_ESTIMATE 関数
構文, 256
INDEX FOR 句
DESCRIBE 文, 704
INDEX ONLY 句
FROM 句, 766
INDEX 句
ALTER INDEX 文, 422
ALTER TABLE 文, 469
CREATE TABLE 文, 655
FROM 句, 766
REORGANIZE TABLE 文, 877
VALIDATE 文, 983
INDICATOR 句
GET DESCRIPTOR 文 [ESQL], 773
SET DESCRIPTOR 文 [ESQL], 907
IN EXCLUSIVE MODE 句
LOCK TABLE 文, 831
INLINE 句
ALTER TABLE 文, 468
CREATE TABLE 文, 654
INNER JOIN 句
FROM 句 SQL 構文, 758
INOUT 句
CREATE PROCEDURE 文, 603
INPUT INTO 文
Interactive SQL 構文, 794
INPUT 句
DESCRIBE 文, 701
INPUT 文
Interactive SQL 構文, 794
構文, 794
ストアドプロシージャーでは使用できない, 800
INSENSITIVE 句
DECLARE CURSOR 文, 687

FOR 文, 754
INSERT INTO 文
構文, 802
INSERT permission 句
GRANT 文, 784
INSERTSTR 関数
構文, 256
INSERT 句
REVOKE 文, 887
INSERT 句
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
CREATE TRIGGER 文, 671
INSERT パーミッション
REVOKE 文, 888
INSERT パーミッション
GRANT 文, 782
INSERT 文
構文, 802
データベースオプションの設定, 804
ビューの更新, 805
IN SHARE MODE 句
LOCK TABLE 文, 831
INSTALL EXTERNAL OBJECT 文
構文, 807
INSTALL JAVA 文
JAVA クラスのインストール, 809
構文, 809
INSTEAD OF 句
CREATE TRIGGER 文, 670
INSTEAD OF トリガー
dropped by CREATE OR REPLACE VIEW によって削除, 681
CREATE TRIGGER 文, 670
ビューの置き換え, 681
ビューの作成, 681
ビューの変更, 485
INTEGER データ型
構文, 99
日付と時刻の比較, 130
INTEGRATED LOGIN 句
REVOKE 文, 887
[Interactive SQL]
BYE SQL 構文, 746
CLEAR 文の構文, 510
CONFIGURE SQL 構文, 516
CONNECT SQL 構文, 516
DESCRIBE CONNECTION 構文, 704
DESCRIBE SQL 構文, 704
DISCONNECT SQL 構文, 707
EXIT SQL 構文, 746
HELP SQL 構文, 791
INPUT SQL 構文, 794
INPUT 文のエンコードの指定, 796
OUTPUT 文, 848
OUTPUT 文のエンコードの指定, 849
PARAMETERS SQL 構文, 854
QUIT SQL 構文, 746
READ SQL 構文, 863
READ 文のコードの指定, 864
SET CONNECTION SQL 構文, 906
SET OPTION SQL 構文, 916
START DATABASE 文, 935
START LOGGING 構文, 940
START SERVER 構文, 938
STOP LOGGING 構文, 949
SYSTEM SQL 構文, 959
サポート対象外の RESUME 文, 882
プロシージャープロファイリング, 1170
文インジケーター, 406
リターンコード, 746
INTERSECT ALL 句
INTERSECT ぶん, 810
INTERSECT DISTINCT 句
INTERSECT ぶん, 810
INTERSECT 句
INTERSECT ぶん, 810
データベースオプションの設定, 811
INTERSECT 文
構文, 810
Interval イベント条件
説明, 222
INTO CLIENT FILE 句
UNLOAD 文, 966
INTO DESCRIPTOR 句
PUT 文 [ESQL], 860
INTO FILE 句
UNLOAD 文, 966
INTO LOCAL TEMPORARY TABLE 句
SELECT 文, 899
INTO VARIABLE 句
UNLOAD 文, 967
INTO 句
EXECUTE 文, 743
EXPLAIN 文 [ESQL], 747
FETCH 文, 749

GET DATA 文, 771
GET OPTION 文 [ESQL], 774
INPUT 文, 799
MERGE 文, 835
OUTPUT 文, 848
PUT 文 [ESQL], 860
SELECT 文, 899
INTTOHEX 関数
構文, 257
IN 句
CREATE FUNCTION 文, 563
CREATE FUNCTION 文 [Web サービス], 555
CREATE MATERIALIZED VIEW 文, 576
CREATE PROCEDURE 文, 603
CREATE TABLE 文, 653
CREATE TEXT INDEX 文, 668
LOCK TABLE 文, 831
IN 探索条件
構文, 50
IOParallelism プロパティ
DB_EXTENDED_PROPERTY 関数, 203
IPAddressMonitorPeriod プロパティ
sa_server_option での設定, 1179
IS
3 値的論理, 59
論理演算子の説明, 9
ISDATE 関数
構文, 258
IS DISTINCT FROM 探索条件
構文, 39
ISENCRYPTED 関数
構文, 259
IS FALSE 探索条件
構文, 58
IS NOT DISTINCT FROM 探索条件
構文, 39
IS NOT NULL 探索条件
構文, 57
IS NOT OF 式
探索条件, 34
ISNULL 関数
構文, 260
IS NULL 探索条件
構文, 57
ISNUMERIC 関数
構文, 261
ISO 8601
サポート, 106
IS OF type-expression
探索条件, 36
IS OF 式
探索条件, 34
isolation_level オプション
DELETE 文に設定, 699
EXCEPT 文に設定, 738
INSERT 文に設定, 804
INTERSECT 文に設定, 811
MERGE 文での上書き, 839
SELECT 文での上書き, 903
UNION 文に設定, 964
UPDATE 文に設定, 979
ISOLATION LEVEL 句
OPEN 文, 845
IS TRUE 探索条件
構文, 58
IS UNKNOWN 探索条件
構文, 58
ISYSARTICLE
システムテーブル, 994
ISYSARTICLECOL
システムテーブル, 994
ISYSATTRIBUTE
システムテーブル, 994
ISYSATTRIBUTENAME
システムテーブル, 994
ISYSCAPABILITY
システムテーブル, 994
ISYSCHECK
システムテーブル, 994
ISYSCOLPERM
システムテーブル, 995
ISYSCOLSTAT
システムテーブル, 995
データベースが暗号化されている場合の暗号化, 995
テーブルの暗号化が有効になっている場合の暗号化, 995
統計のロード, 814
ISYSCONSTRAINT
システムテーブル, 995
ISYSDBFILE
システムテーブル, 995
ISYSDBSPACE
システムテーブル, 996
ISYSDBSPACEPERM
システムテーブル, 996

ISYSDEPENDENCY
システムテーブル, 995
ISYSDOMAIN
システムテーブル, 996
ISYSEVENT
システムテーブル, 996
ISYSEXTERNLOGIN
システムテーブル, 996
データベースが暗号化されている場合の暗号化, 996
テーブルの暗号化が有効になっている場合の暗号化, 996
ISYSFKEY
システムテーブル, 997
ISYSGROUP
システムテーブル, 997
ISYSHISTORY
システムテーブル, 997
ISYSIDX
システムテーブル, 997
ISYSIDXCOL
システムテーブル, 998
ISYSJAR
システムテーブル, 998
ISYSJARCOMPONENT
システムテーブル, 998
ISYSJAVACLASS
システムテーブル, 998
ISYSLOGINMAP
システムテーブル, 998
ISYSLOGINPOLICY
システムテーブル, 999
ISYSLOGINPOLICYOPTION
システムテーブル, 999
ISYSMIRROROPTION
システムテーブル, 999
ISYSMIRRORSERVER
システムテーブル, 999
ISYSMIRRORSERVEROPTION
システムテーブル, 999
ISYSMVOPTION
システムテーブル, 1000
ISYSMVOPTIONNAME
システムテーブル, 1000
ISYSOBJECT
システムテーブル, 1000
ISYSOPTION
システムテーブル, 1000
ISYSOPTSTAT
システムテーブル, 1000
ISYSPHYSIDX
システムテーブル, 1001
ISYSPROCEDURE
システムテーブル, 1001
ISYSPROCPARM
システムテーブル, 1001
ISYSPROCPERM
システムテーブル, 1001
ISYSPROXYTAB
システムテーブル, 1001
ISYSPUBLICATION
システムテーブル, 1002
ISYSREMARK
システムテーブル, 1002
ISYSREMOTEOPTION
システムテーブル, 1002
ISYSREMOTEOPTIONTYPE
システムテーブル, 1002
ISYSREMOTETYPE
システムテーブル, 1002
ISYSREMOTEUSER
システムテーブル, 1003
ISYSSCHEDULE
システムテーブル, 1003
ISYSSEQUENCE
システムテーブル, 1003
ISYSSEQUENCEPERM
システムテーブル, 1003
ISYSSERVER
コンポーネント統合サービスのリモートサーバー, 618
サーバーの追加, 618
システムテーブル, 1004
ISYSSOURCE
システムテーブル, 1004
ISYSSPATIALREFERENCESYSTEM
システムテーブル, 1004
ISYSSQLSERVERTYPE
システムテーブル, 1004
ISYSSUBSCRIPTION
システムテーブル, 1004
ISYSSYNC
システムテーブル, 1005
ISYSSYNCPROFILE
システムテーブル, 1005
ISYSSYNCSCRIPT

システムテーブル, 1005
ISYSTAB
システムテーブル, 1005
ISYSTABCOL
システムテーブル, 1005
ISYSTABLEPERM
システムテーブル, 1006
ISYSTEXTCONFIG
システムテーブル, 1006
ISYSTEXTIDX
システムテーブル, 1006
ISYSTEXTIDXTAB
システムテーブル, 1006
ISYSTRIGGER
システムテーブル, 1006
ISYSTYPEMAP
システムテーブル, 1007
ISYSUNITOFMEASURE
システムテーブル, 1007
ISYSUSER
システムテーブル, 1007
データベースが暗号化されている場合の暗号化, 1007
テーブルの暗号化が有効になっている場合の暗号化, 1007
ISYSUSERAUTHORITY
システムテーブル, 1007
ISYSUSERMESSAGE
システムテーブル, 1007
ISYSUSERTYPE
システムテーブル, 1008
ISYSVIEW
システムテーブル, 1008
ISYSWEBSERVICE
CREATE SERVICE 文 [HTTP Web サービス] を使用して HTTP サービスを追加, 622
HTTP web サービスの変更, 441
HTTP を介した SOAP サービスの追加, 628
HTTP を介した SOAP サービスの変更, 446
システムテーブル, 1008
WITH NULLS NOT DISTINCT 句
CREATE INDEX 文, 569

J

JAR 句
INSTALL JAVA 文, 809
REMOVE JAVA 文, 876
JAR ファイル
COMMENT 文を使用してコメントを追加, 512
INSTALL JAVA 文, 809
削除, 876
Java
Java と SQL の変換, 136
インストール, 809
システムテーブル, 1027
ユーザー定義関数, 144
JavaScript Object Notation
CREATE SERVICE 文 [HTTP Web サービス], 441, 623
Java VM
STOP JAVA 文, 948
Java から SQL のデータ型変換
説明, 136
Java クラス
データベースにロードされた, 1107
トラブルシューティング, 1107
Java クラス
COMMENT 文を使用してコメントを追加, 512
Java データ型
SQL からの変換, 137
SQL への変換, 136
Java と SQL のデータ型変換
説明, 136
jConnect
CREATE DATABASE 文, 525
JCONNECT 句
ALTER DATABASE 文, 409
CREATE DATABASE 文, 525
JDBC
Java から SQL のデータ型変換, 136
SQL から Java のデータ型変換, 137
データ型変換, 136
データベースコンポーネントのアップグレード, 408
JSON
CREATE SERVICE 文 [HTTP Web サービス], 441, 623

K

Kerberos
COMMENT 文を使用してコメントを追加, 512
KERBEROS LOGIN の取り消し, 887
Kerberos ログインの付与, 783
プリンシパルの大文字と小文字の区別, 786

KERBEROS LOGIN 句
REVOKE 文, 887
Kerberos ログイン
GRANT 文, 783
KEY JOIN 句
FROM 句 SQL 構文, 758
KEY 句
ALTER DATABASE 文, 410
CREATE DECRYPTED DATABASE 文, 530
CREATE DECRYPTED FILE 文, 532
CREATE ENCRYPTED DATABASE 文, 535
CREATE ENCRYPTED FILE 文, 538
CREATE ENCRYPTED TABLE DATABASE 文, 535
DROP DATABASE 文, 709
RESTORE DATABASE 文, 880
START DATABASE 文, 935

L

LANGUAGE C_ESQL32 句
CREATE FUNCTION 文 [外部呼び出し], 552
CREATE PROCEDURE 文 [外部呼び出し], 588
LANGUAGE C_ESQL64 句
CREATE FUNCTION 文 [外部呼び出し], 552
CREATE PROCEDURE 文 [外部呼び出し], 588
LANGUAGE C_ODBC32 句
CREATE FUNCTION 文 [外部呼び出し], 552
CREATE PROCEDURE 文 [外部呼び出し], 588
LANGUAGE C_ODBC64 句
CREATE FUNCTION 文 [外部呼び出し], 552
CREATE PROCEDURE 文 [外部呼び出し], 588
LANGUAGE CLR 句
CREATE FUNCTION 文 [外部呼び出し], 552
CREATE PROCEDURE 文 [外部呼び出し], 588
LANGUAGE JAVA 句
CREATE FUNCTION 文 [外部呼び出し], 553
CREATE PROCEDURE 文 [外部呼び出し], 589
LANGUAGE PERL 句
CREATE FUNCTION 文 [外部呼び出し], 553
CREATE PROCEDURE 文 [外部呼び出し], 588
LANGUAGE PHP 句
CREATE FUNCTION 文 [外部呼び出し], 553
CREATE PROCEDURE 文 [外部プロシージャー], 589
LAST_VALUE 関数
構文, 262
LAST USER 句
CREATE TABLE 文, 651
LAST USER デフォルト
CREATE TABLE 文, 467, 658
LAST USER 特別値
構文, 67
LAST 句
FETCH 文, 750
LCASE 関数
構文, 264
LEAVE 文
構文, 812
LEFT OUTER JOIN 句
FROM 句 SQL 構文, 758
LEFT 関数
構文, 265
LENGTH 関数
構文, 266
LENGTH 句
GET DESCRIPTOR 文 [ESQL], 773
SET DESCRIPTOR 文 [ESQL], 907
LEN 関数
構文, 266
LESSER 関数
構文, 267
LIKE 探索条件
REGEXP と SIMILAR TO との比較, 40
大文字と小文字の区別, 43
構文, 42
照合, 43
パターン長, 43
LIKE 探索条件、REGEXP 探索条件、SIMILAR TO 探索条件
説明, 40
LIMIT 句
ATTACH TRACING 文, 487
SELECT 文, 898
LINEAR UNIT OF MEASURE 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
LIST 関数
構文, 268
LivenessTimeout プロパティ
sa_server_option での設定, 1179
LOAD STATISTICS 文
構文, 814
LOAD TABLE 文

構文, 814
データベースミラーリングでの使用, 826
LOCAL DATABASE 句
ATTACH TRACING 文, 487
LOCATE 関数
構文, 271
Location
HTTP_RESPONSE_HEADER 関数, 251
LOCATION 句
ALTER EXTERNAL ENVIRONMENT 文, 420
locked オプション
ログインポリシー, 573
LOCK FEATURE 文
構文, 830
LOCK TABLE 文
構文, 831
LOG10 関数
構文, 273
LogDiskSpace システムイベント
例, 223
logfile オプション
ALTER MIRROR SERVER 文, 432
CREATE MIRROR SERVER 文, 581
LogFreePercent イベント条件
説明, 222
LogFreeSpace イベント条件
説明, 222
LOGIN POLICY
ALTER USER 文, 482
LOGIN POLICY 句
CREATE USER 文, 677
LogSize イベント条件
説明, 222
LOG 関数
構文, 272
LONG BINARY データ型
構文, 123
LONG NAMES 句
DESCRIBE 文, 702
LONG NVARCHAR データ型
構文, 87
説明, 87
LONG VARBIT データ型
構文, 104
LONG VARCHAR データ型
構文, 88
LOOP 句
LOOP 文, 832
LOOP 文
構文, 832
LOWER 関数
構文, 273
LTRIM 関数
構文, 274

M

MANUAL REFRESH 句
ALTER MATERIALIZED VIEW 文, 428, 576
CREATE TEXT INDEX 文, 668
MAPI
MAPI システムプロシージャーと SMTP システムプロシージャー, 1243
MAPI システムプロシージャーと SMTP システムプロシージャーのリターンコード, 1031
拡張システムプロシージャー, 1030
電子メールセッションの開始, 1246
電子メールセッションの終了, 1249
MAPI システムプロシージャーと SMTP システムプロシージャー
リターンコード, 1031
MATCH FULL 句
CREATE TABLE 文, 660
MATCH SIMPLE 句
CREATE TABLE 文, 660
MATCH UNIQUE FULL 句
CREATE TABLE 文, 660
MATCH UNIQUE SIMPLE 句
CREATE TABLE 文, 660
MATCH 句
ALTER TABLE 文, 468
CREATE TABLE 文, 660
materialized_view_optimization オプション
DELETE 文に設定, 699
EXCEPT 文に設定, 738
INSERT 文に設定, 804
INTERSECT 句に設定, 811
INTERSECT 文に設定, 811
SELECT 文での上書き, 902
UNION 文に設定, 964
UPDATE 文に設定, 979
materialized-view-name
一般的な SQL 構文要素, 403
MATERIALIZED VIEW OPTIMIZATION オプション
MERGE 文, 839

MATERIALIZED VIEW OPTIMIZATION 句
DELETE 文, 699
EXCEPT 文, 738
INSERT 文, 804
INTERSECT 文, 811
SELECT 文, 902
UPDATE 文, 979
MATERIALIZED VIEW 句
TRUNCATE 文, 961
VALIDATE 文, 983
max_connections オプション
ログインポリシー, 573
max_days_since_login オプション
ログインポリシー, 573
max_disconnected_time オプション
SET MIRROR OPTION 文, 909
max_failed_login_attempts オプション
ログインポリシー, 573
max_non_dba_connections オプション
ログインポリシー, 573
max_query_tasks オプション
DELETE 文に設定, 699
EXCEPT 文に設定, 738
INSERT 文に設定, 804
INTERSECT 文に設定, 811
MERGE 文での上書き, 839
SELECT 文での上書き, 903
UNION 文に設定, 964
UPDATE 文に設定, 979
max_retry_connect_time オプション
SET MIRROR OPTION 文, 909
MAXIMUM TERM LENGTH 句
ALTER TEXT CONFIGURATION 文, 477
MaxMultiProgrammingLevel プロパティ
sa_server_option での設定, 1179
MAXVALUE 句
ALTER SEQUENCE 文, 438
CREATE SEQUENCE 文, 617
MAX WRITE 句
BACKUP 文, 492
MAX 関数
構文, 275
MEDIAN 関数
構文, 276
MEMBERSHIP IN GROUP 句
REVOKE 文, 887
MEMBERSHIP IN GROUP 句
GRANT 文, 784
MERGE 句
ALTER SYNCHRONIZATION PROFILE 文, 457
MERGE 文
構文, 834
MessageCategoryLimit プロパティ
sa_server_option での設定, 1179
MESSAGE 文
構文, 841
METHODS 句
ALTER SERVICE 文 [HTTP Web サービス], 441
ALTER SERVICE 文 [SOAP Web サービス], 446
CREATE SERVICE 文 [HTTP Web サービス], 443, 625
CREATE SERVICE 文 [SOAP Web サービス], 450, 632
MIME base64
データのエンコード, 159
データの復号化, 158
MINIMUM TERM LENGTH 句
ALTER TEXT CONFIGURATION 文, 477
MinMultiProgrammingLevel プロパティ
sa_server_option での設定, 1179
MINUTES 関数
構文, 280
MINUTE 関数
構文, 280
MINVALUE 句
ALTER SEQUENCE 文, 438
CREATE SEQUENCE 文, 617
MIN 関数
構文, 278
MirrorServerState プロパティ
DB_EXTENDED_PROPERTY 関数, 203
MirrorState プロパティ
DB_EXTENDED_PROPERTY 関数, 203
MIRROR 句
CREATE DATABASE 文, 526
Mobile Link
ALTER PUBLICATION 文, 435
ALTER SYNCHRONIZATION PROFILE 文, 457
ALTER SYNCHRONIZATION SUBSCRIPTION 文, 458
ALTER SYNCHRONIZATION USER 文, 461
CREATE PUBLICATION 文, 610
CREATE SYNCHRONIZATION PROFILE 文, 646

CREATE SYNCHRONIZATION SUBSCRIPTION 文, 647
CREATE SYNCHRONIZATION USER 文, 650
DROP PUBLICATION 文, 721
DROP SYNCHRONIZATION PROFILE 文, 729
DROP SYNCHRONIZATION SUBSCRIPTION 文, 729
START SYNCHRONIZATION DELETE 文, 943
START SYNCHRONIZATION SCHEMA CHANGE 文, 944
STOP SYNCHRONIZATION DELETE 文, 951
STOP SYNCHRONIZATION SCHEMA CHANGE 文, 957
Mobile Link ユーザー
ALTER SYNCHRONIZATION USER 文, 461
CREATE SYNCHRONIZATION USER 文, 650
DROP SYNCHRONIZATION USER 文, 731
MOD 関数
構文, 282
MONEY データ型
構文, 103
MONTHNAME 関数
構文, 283
MONTHS 関数
構文, 284
MONTH 関数
構文, 283

N

NAMESPACE 句
CREATE FUNCTION 文 [Web クライアント], 558
CREATE PROCEDURE 文 [Web クライアント], 598
NAME 句
GET DESCRIPTOR 文 [ESQL], 773
NATURAL JOIN 句
FROM 句 SQL 構文, 758
NCHAR COLLATION 句
CREATE DATABASE 文, 526
照合の適合化, 526
NcharCollation プロパティ
DB_EXTENDED_PROPERTY 関数, 203
NCHAR 関数
構文, 286
NCHAR データ型
CHAR データ型との比較, 129
LIKE 探索条件での使用, 46
NCHAR カラムに DESCRIBE を使用, 88
REGEXP 探索条件での使用, 47
SIMILAR TO 探索条件での使用, 49
構文, 88
説明, 88
NEAR 句
CONTAINS 探索条件, 51
NEWID 関数
構文, 286
NEW SUBSCRIBE BY 句
UPDATE 文, 976
UPDATE 文 [SQL Remote], 973
NEW 句
INSTALL EXTERNAL OBJECT 文, 807
INSTALL JAVA 文, 809
NEXT_CONNECTION 関数
構文, 288
NEXT_DATABASE 関数
構文, 289
NEXT_HTTP_HEADER 関数
構文, 290
NEXT_HTTP_RESPONSE_HEADER 関数
構文, 291
NEXT_HTTP_VARIABLE 関数
構文, 292
NEXT_SOAP_HEADER 関数
構文, 293
NextScheduleTime プロパティ
DB_EXTENDED_PROPERTY 関数, 203
NEXT 句
FETCH 文, 750
N-gram
ALTER TEXT CONFIGURATION 文, 477
NO COPY 句
BACKUP 文, 492
NO INDEX 句
ALTER TABLE 文, 469
CREATE TABLE 文, 655
FROM 句, 766
NOLOCK テーブルヒント
FROM 句, 764
NONCLUSTERED 句
ALTER INDEX 文, 422
NO RESULT SET 句
CREATE PROCEDURE 文, 605
CREATE PROCEDURE 文 [T-SQL], 600
CREATE PROCEDURE 文 [外部呼び出し], 586

NO SCROLL カーソル
宣言, 686
NO SCROLL 句
DECLARE CURSOR 文, 687
FOR 文, 753
NOSTRIP 句
INPUT 文, 799
NOT
3 値的論理, 59
ビット処理演算子, 11
論理演算子の説明, 9
NOT COMPRESSED 句
LOAD TABLE 文, 819
NOT DETERMINISTIC 句
CREATE FUNCTION 文 [外部プロシージャー], 551
CREATE FUNCTION 文 [外部呼び出し], 550
CREATE FUNCTION 文 [ユーザー定義], 564
NOT ENCRYPTED 句
ALTER MATERIALIZED VIEW 文, 428
LOAD TABLE 文, 820
NOTFOUND 句
WHENEVER 文 [ESQL], 987
NOT NULL 句
ALTER TABLE 文, 464
CREATE EXISTING TABLE 文, 546
CREATE TABLE 文, 655
NOT TRANSACTIONAL 句
CREATE LOCAL TEMPORARY TABLE 文, 572
CREATE TABLE 文, 653
DECLARE LOCAL TEMPORARY TABLE 文, 691
NOW 関数
構文, 294
NTEXT データ型
構文, 89
NULL
3 値的論理, 68
ASE 互換性, 69
ISNULL 関数, 260
NULL 値, 67
NULL 値に割り付けられた領域, 68
NULL 引数が指定されている場合に関数から返される, 139
説明, 67
NULLABLE 句
GET DESCRIPTOR 文 [ESQL], 773
NULLIF 関数
CASE 式での使用, 17
説明, 295
NULL 句
ALTER TABLE 文, 464
CASE 文, 505
CASE 文 [T-SQL], 507
CREATE DOMAIN 文, 533
CREATE TABLE 文, 655
NULL 値
ドメイン, 533
NULL 定数
NUMERIC への変換, 134
文字列型への変換, 134
NULL 定数を NUMERIC 型と文字列型に変換する
説明, 134
number
一般的な SQL 構文要素, 403
NUMBER 関数
SQL Remote 更新, 974
構文, 296
NUMERIC データ型
DOUBLE からの変換, 135
構文, 99
日付と時刻の比較, 130
NVARCHAR データ型
NVARCHAR カラムに DESCRIBE を使用, 89
構文, 89
説明, 89

O

OBJECT_ID 関数
構文, 150
OBJECT_NAME 関数
構文, 150
ODBC
静的カーソルの宣言, 686
OFFSET 句
GET DATA 文, 771
SELECT 文, 898
OLAP
CUBE 演算, 789
GROUP BY 句, 788
GROUPING SETS 演算, 788
GROUPING 関数, 238
ROLLUP 演算, 789
WINDOW 句, 989

OLAP 関数
AVG 関数, 156
COUNT_BIG 関数, 185
COUNT 関数, 183
COVAR_POP 関数, 187
CUME_DIST 関数, 191
DENSE_RANK 関数, 213
MAX 関数, 275
MEDIAN 関数, 276
MIN 関数, 278
PERCENT_RANK 関数, 300
RANK 関数, 311
REGR_AVGX 関数, 316
REGR_AVGY 関数, 317
REGR_COUNT 関数, 318
REGR_INTERCEPT 関数, 320
REGR_R2 関数, 321
REGR_SLOPE 関数, 322
REGR_SXX 関数, 324
REGR_SXY 関数, 326
ROW_NUMBER 関数, 336
STDDEV_POP 関数, 354
STDDEV_SAMP 関数, 356
STDDEV 関数, 354
SUM 関数, 363
VAR_POP 関数, 384
VAR_SAMP 関数, 386
OLD KEY 句
CREATE ENCRYPTED FILE 文, 538
CREATE ENCRYPTED TABLE DATABASE 文, 535
OLD SUBSCRIBE BY 句
UPDATE 文, 976
UPDATE 文 [SQL Remote], 973
on_tsql_error オプション
ON EXCEPTION RESUME 句, 605
ON COMMIT 句
CREATE LOCAL TEMPORARY TABLE 文, 572
CREATE TABLE 文, 653
DECLARE LOCAL TEMPORARY TABLE 文, 691
ON DATABASE 句
STOP DATABASE 文, 946
ON EXCEPTION RESUME 句
CREATE FUNCTION 文 [ユーザー定義], 564
CREATE PROCEDURE 文, 605
説明, 605
ON EXISTING ERROR 句
BACKUP 文, 491
DEFAULT カラムでの動作, 804
ON EXISTING SKIP 句
DEFAULT カラムでの動作, 804
ON EXISTING 句
INSERT 文, 803
ON SUBSCRIPTION 句
START SYNCHRONIZATION SCHEMA CHANGE 文, 944
ON 句
REVOKE 文, 887
ON 句
ALTER STATISTICS 文, 456
ALTER TRIGGER 文, 481
CREATE EVENT 文, 543
CREATE INDEX 文, 567
CREATE TEXT INDEX 文, 668
CREATE TRIGGER [Transact-SQL] 文, 676
DROP STATISTICS 文, 727
DROP TEXT INDEX 文, 733
MERGE 文, 836
START DATABASE 文, 935
TRUNCATE TEXT INDEX 文, 963
VALIDATE 文, 983
ON フレーズ
探索条件, 34
OPENSTRING 句
FROM 句, 762
例, 770
openxml システムプロシージャー
dbo は所有者ではない, 1029
構文, 1035
サポートするテストの種類, 1038
サポートするメタプロパティのリスト, 1036
OPEN 文
Embedded SQL 構文, 844
WITH HOLD カーソル, 844
構文, 844
optimization_goal オプション
DELETE 文に設定, 699
EXCEPT 文に設定, 738
INSERT 文に設定, 804
INTERSECT 文に設定, 811
MERGE 文での上書き, 839
SELECT 文での上書き, 903
UNION 文に設定, 964
UPDATE 文に設定, 979
optimization_level オプション

DELETE 文に設定, 699
EXCEPT 文に設定, 738
INSERT 文に設定, 804
INTERSECT 文に設定, 811
MERGE 文での上書き, 839
SELECT 文での上書き, 903
UNION 文に設定, 964
UPDATE 文に設定, 979
optimization_workload オプション
DELETE 文に設定, 699
EXCEPT 文に設定, 738
INSERT 文に設定, 804
INTERSECT 文に設定, 811
MERGE 文での上書き, 839
SELECT 文での上書き, 903
UNION 文に設定, 964
UPDATE 文に設定, 979
OptionWatchAction プロパティ
sa_server_option での設定, 1179
OptionWatchList プロパティ
sa_server_option での設定, 1179
OPTION 句
CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 648
CREATE SYNCHRONIZATION USER, 650
DELETE 文, 699
EXCEPT 文, 738
INSERT 文, 804
INTERSECT 文, 811
MERGE 文, 839
SELECT 文, 902
UNION 文, 964
UPDATE 文, 979
OR
3 値的論理, 59
ビット処理演算子, 11
論理演算子の説明, 9
Oracle データベース
sa_migrate システムプロシージャーを使用して SQL Anywhere にマイグレート, 1128
ORDER BY 句
DELETE 文, 699
EXCEPT 文, 738
INTERSECT 文, 810
SELECT 文, 895
UPDATE 文, 979
UPDATE 文 [SQL Remote], 973
WINDOW 句, 990
説明, 900
ORDER 句
CREATE TRIGGER 文, 671
LOAD TABLE 文, 823
UNLOAD 文, 968
ORGANIZATION 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
OR REPLACE 句
CREATE FUNCTION 文 [Web サービス], 555
CREATE FUNCTION 文 [外部プロシージャー], 550
CREATE FUNCTION 文 [外部呼び出し], 550
CREATE FUNCTION 文 [ユーザー定義], 563
CREATE MIRROR SERVER 文, 580
CREATE PROCEDURE 文, 602
CREATE PROCEDURE 文 [T-SQL], 599
CREATE PROCEDURE 文 [Web クライアント], 592
CREATE PROCEDURE 文 [外部呼び出し], 583
CREATE SEQUENCE 文, 617
CREATE SPATIAL REFERENCE SYSTEM 文, 635
CREATE SYNCHRONIZATION PROFILE 文, 646
CREATE TRIGGER 文, 670
CREATE VARIABLE 文, 679
CREATE VIEW 文, 681
OUTER APPLY 句
FROM 句, 763
output_log_send_limit
SQL Remote 構文, 919
output_log_send_now
SQL Remote 構文, 919
output_log_send_on_error
SQL Remote 構文, 919
OUTPUT 句
CREATE PROCEDURE 文 [T-SQL], 599
DESCRIBE 文, 701
OUTPUT 文
構文, 848
OUT 句
CREATE PROCEDURE 文, 603
owner
一般的な SQL 構文要素, 403
OWNER.TABLE.COLUMN 句

DESCRIBE 文 [ESQL], 700
OwnerName プロパティ
sa_materialized_view_info システムプロシージャー, 1119

P

page_timeout オプション
SET MIRROR OPTION 文, 909
PAGE SIZE 句
CREATE DATABASE 文, 526
PARAMETERS 文
Interactive SQL 構文, 854
PARTITION BY 句
WINDOW 句, 989
PASSTHROUGH FOR SUBSCRIPTION 句
PASSTHROUGH 文 [SQL Remote], 855
PASSTHROUGH FOR 句
PASSTHROUGH 文 [SQL Remote], 855
PASSTHROUGH ONLY 句
PASSTHROUGH 文 [SQL Remote], 855
PASSTHROUGH STOP 句
PASSTHROUGH 文 [SQL Remote], 855
PASSTHROUGH 文
SQL Remote 構文, 855
password_expiry_on_next_login オプション
ログインポリシー, 573
password_grace_time オプション
ログインポリシー, 573
password_life_time オプション
ログインポリシー, 573
PATINDEX 関数
構文, 297
PCTFREE 句
CREATE LOCAL TEMPORARY TABLE 文, 571
CREATE TABLE 文, 662
DECLARE LOCAL TEMPORARY TABLE 文, 691
LOAD TABLE, 823
PCTFREE 設定
ALTER TABLE 文, 462
CREATE LOCAL TEMPORARY TABLE 構文, 571
CREATE TABLE 文, 651
DECLARE LOCAL TEMPORARY TABLE 構文, 691
LOAD TABLE 構文, 814
PERCENT_RANK 関数
構文, 300
PERL 句
INSTALL EXTERNAL OBJECT 文, 807
PHP 句
INSTALL EXTERNAL OBJECT 文, 807
PI 関数
構文, 301
PLAN 関数
構文, 302
POLYGON FORMAT 句
ALTER SPATIAL REFERENCE SYSTEM 文, 455
CREATE SPATIAL REFERENCE SYSTEM 文, 640
POWER 関数
構文, 303
PRECEDING 句
WINDOW 句, 990
PRECISION 句
GET DESCRIPTOR 文 [ESQL], 773
SET DESCRIPTOR 文 [ESQL], 907
preferred オプション
ALTER MIRROR SERVER 文, 432
CREATE MIRROR SERVER 文, 581
PREFILTER EXTERNAL NAME 句
ALTER TEXT CONFIGURATION 文, 478
PREFIX 句
ALTER TABLE 文, 468
CREATE TABLE 文, 654
PREPARE TO COMMIT 文
構文, 858
PREPARE 文
Embedded SQL 構文, 856
PRESERVE 句
CREATE LOCAL TEMPORARY TABLE 文, 571
DECLARE LOCAL TEMPORARY TABLE 文, 691
PRIMARY KEY 句
ALTER INDEX 文, 422
ALTER TABLE 文, 468
CREATE TABLE 文, 659
REORGANIZE TABLE 文, 877
VALIDATE 文, 983
PRIMARY KEY 制約句
CREATE TABLE 文, 659
PRINT 文
Transact-SQL 構文, 859
PRIOR 句

FETCH 文, 750
ProcedureProfiling プロパティ
sa_server_option での設定, 1179
PROCEDURE 句
ALTER DATABASE 文, 409
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
DESCRIBE 文, 704
PROFILE authority 句
GRANT 文, 783
ProfileFilterConn プロパティ
sa_server_option での設定, 1180
ProfileFilterUser プロパティ
sa_server_option での設定, 1180
PROFILE 句
REVOKE 文, 887
PROFILE 権限
REVOKE 文, 888
PROFILE 権限
GRANT 文, 782
PROMPT 句
INPUT 文, 799
Properties プロパティ
DB_EXTENDED_PROPERTY 関数, 203
PROPERTY_DESCRIPTION 関数
構文, 303
PROPERTY_NAME 関数
構文, 307
PROPERTY_NUMBER 関数
構文, 308
PROPERTY 関数
構文, 304
PROXY 句
CREATE FUNCTION 文 [Web クライアント], 559
CREATE PROCEDURE 文 [Web クライアント], 595
PUBLICATION 句
UPDATE 文, 976
UPDATE 文 [SQL Remote], 973
PUBLIC 句
SET OPTION 文, 913
PUBLISH パーミッション
SQL Remote の取り消し, 885
付与, 777
PunctuationSensitivity プロパティ
DB_EXTENDED_PROPERTY 関数, 203
PURGE 句
FETCH 文, 751
PUT 文
Embedded SQL 構文, 860

Q

QUARTER 関数
構文, 308
query block
一般的な SQL 構文要素, 403
query-block
一般的な SQL 構文要素, 403
query-expression
一般的な SQL 構文要素, 403
QuittingTime プロパティ
sa_server_option での設定, 1180
QUIT 文
Interactive SQL 構文, 746
quoted_identifier オプション
Transact-SQL SET 文を使用して設定, 929
T-SQL 式の互換性, 34
QUOTES 句
LOAD TABLE 文, 823
UNLOAD 文, 968
QUOTE 句
LOAD TABLE 文, 823
OUTPUT 文, 851
UNLOAD 文, 968

R

RADIANS 関数
構文, 309
RAISERROR 句
MERGE 文, 837
RAISERROR 文
構文, 862
RAND 関数
構文, 310
RANGE 句
WINDOW 句, 990
RANK 関数
構文, 311
RAW
CREATE SERVICE 文 [HTTP Web サービス], 441, 623
READ_CLIENT_FILE 関数
構文, 313
READCLIENTFILE authority 句

GRANT 文, 783
READCLIENTFILE 権限
REVOKE 文, 888
READCLIENTFILE 権限
GRANT 文, 782
READ COMMITTED 句
REFRESH MATERIALIZED VIEW 文, 867
READCOMMITTED テーブルヒント
FROM 句, 764
READFILE authority 句
GRANT 文, 783
READFILE 権限
REVOKE 文, 888
READFILE 権限
GRANT 文, 782
READONLY 句
CREATE SERVER 文, 620
READPAST テーブルヒント
FROM 句, 764
ReadPK ロック
sa_locks システムプロシージャー, 1112
READTEXT 文
Transact-SQL 構文, 866
READ UNCOMMITTED 句
REFRESH MATERIALIZED VIEW 文, 867
READUNCOMMITTED テーブルヒント
FROM 句, 764
READ 文
構文, 863
REAL データ型
構文, 101
日付と時刻の比較, 130
REBUILD 句
ALTER INDEX 文, 423
RECOMPILE 句
ALTER FUNCTION 文, 421
ALTER PROCEDURE 文, 433
ALTER VIEW 文, 485
RECOVER 句
BACKUP 文, 492
REFERENCES permission 句
GRANT 文, 784
REFERENCES 句
REVOKE 文, 887
REFERENCES 句
ALTER TABLE 文, 468
CREATE TABLE 文, 660
REFERENCES パーミッション
REVOKE 文, 888
REFERENCES パーミッション
GRANT 文, 782
REFERENCING 句
CREATE TRIGGER 文, 672
Referer
HTTP_HEADER 関数, 249
REFRESH MATERIALIZED VIEW 文
構文, 867
REFRESH TEXT INDEX 文
構文, 870
REFRESH TRACING LEVELS 文
sa_diagnostic_tracing_level テーブルにデータを作成する, 1022
REFRESH TRACING LEVEL 文
構文, 872
診断トレーシング, 872
RefreshType プロパティ
sa_materialized_view_info システムプロシージャー, 1119
REFRESH 句
ALTER TEXT INDEX 文, 480
CREATE TEXT INDEX 文, 668
REGEXP_SUBSTR 関数
構文, 314
正規表現, 18
データベースの照合と一致, 314
REGEXP 探索条件
LIKE と SIMILAR TO との比較, 40
構文, 47
正規表現, 18
データベース照合と一致, 47
部分文字クラスの一致, 47
REGR_AVGX 関数
構文, 316
REGR_AVGY 関数
構文, 317
REGR_COUNT 関数
構文, 318
REGR_INTERCEPT 関数
構文, 320
REGR_R2 関数
構文, 321
REGR_SLOPE 関数
構文, 322
REGR_SXX 関数
構文, 324
REGR_SXY 関数

構文, 326
REGR_SYY 関数
構文, 327
RELATIVE 句
FETCH 文, 750
RELEASE SAVEPOINT 文
構文, 873
REMAINDER 関数
構文, 328
RememberLastPlan プロパティ
sa_server_option での設定, 1180
RememberLastStatement プロパティ
sa_server_option での設定, 1180
REMOTE DBA 権限
付与, 778
REMOTE DBA パーミッション
SQL Remote の取り消し, 886
REMOTE LOGIN 句
CREATE EXTERNLOGIN 文, 548
remoteoptiontype ビュー
説明, 1294
remoteoption ビュー
説明, 1293
REMOTE RESET 文
SQL Remote 構文, 874
REMOTE 句
CREATE EVENT 文, 540
REMOTE パーミッション
SQL Remote の取り消し, 887
付与, 780
REMOVE EXTERNAL OBJECT 文
構文, 875
REMOVE JAVA 文
構文, 876
RENAME 句
ALTER DBSPACE 文, 414
ALTER DOMAIN 文, 416
ALTER INDEX 文, 422
ALTER SPATIAL REFERENCE SYSTEM 文, 453
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
ALTER TABLE 文, 472
ALTER TEXT INDEX 文, 480
RESTORE DATABASE 文, 880
REORGANIZE TABLE 文
構文, 877
REPEATABLE READ 句
REFRESH MATERIALIZED VIEW 文, 867
REPEATABLEREAD テーブルヒント
FROM 句, 764
REPEAT 関数
構文, 329
REPLACE 関数
構文, 330
REPLICATE 関数
構文, 331
RequestFilterConn プロパティ
sa_server_option での設定, 1181
RequestFilterDB プロパティ
sa_server_option での設定, 1181
RequestLogFile プロパティ
sa_server_option での設定, 1181
クライアントでの文のキャッシュ, 1181
RequestLogging プロパティ
sa_server_option での設定, 1182
RequestLogMaxSize プロパティ
sa_server_option での設定, 1182
RequestLogNumFiles プロパティ
sa_server_option での設定, 1183
RequestTiming プロパティ
sa_server_option での設定, 1183
RESET LOGIN POLICY 句
ALTER USER 文, 483
RESIGNAL 文
構文, 879
RESOLVE 句
CREATE TRIGGER 文, 670
RESOURCE authority 句
GRANT 文, 784
RESOURCE 句
REVOKE 文, 887
RESOURCE 権限
REVOKE 文, 888
RESOURCE 権限
GRANT 文, 782
RESTART WITH 句
ALTER SEQUENCE 文, 438
RESTORE DATABASE 文
構文, 880
RESTORE DEFAULT CALIBRATION 句
ALTER DATABASE 文, 409
RESULT 句
CREATE FUNCTION 文 [外部呼び出し], 585
CREATE PROCEDURE 文, 604
RESUME 文

Interactive SQL でサポート対象外, 882
構文, 881
RETURNED_LENGTH 句
GET DESCRIPTOR 文 [ESQL], 773
RETURNS 句
CREATE FUNCTION 文, 563
CREATE FUNCTION 文 [Web クライアント], 556
CREATE FUNCTION 文 [外部呼び出し], 550
RETURN 文
構文, 882
REVERSE 関数
構文, 332
REVOKE BACKUP 文
構文, 887
REVOKE CONNECT 文
構文, 887
REVOKE CONSOLIDATE 文
SQL Remote SQL 構文, 884
REVOKE CREATE ON 文
構文, 887
REVOKE DBA 文
構文, 887
REVOKE EXECUTE 句
REVOKE 文, 887
REVOKE GROUP 文
構文, 887
REVOKE INTEGRATED LOGIN 文
構文, 887
REVOKE KERBEROS LOGIN 文
構文, 887
REVOKE MEMBERSHIP IN GROUP 文
構文, 887
REVOKE PROFILE 文
構文, 887
REVOKE PUBLISH 文
SQL Remote SQL 構文, 885
REVOKE REMOTE DBA 文
SQL Remote SQL 構文, 886
REVOKE REMOTE 文
SQL Remote SQL 構文, 887
REVOKE RESOURCE 文
構文, 887
REVOKE VALIDATE 文
構文, 887
REVOKE 文
構文, 887
REWRITE 関数
構文, 332
RIGHT OUTER JOIN 句
FROM 句 SQL 構文, 758
RIGHT 関数
構文, 334
RI 制約
ALTER TABLE 文, 651
ALTER TABLE 文を使用した追加、削除、変更, 462
ALTER TABLE 文を使用した名前変更, 473
基本となるインデックスの名前が変更されたときに名前変更されない, 423
role-name
一般的な SQL 構文要素, 403
ROLLBACK TO SAVEPOINT 文
構文, 891
ROLLBACK TRANSACTION 文
Transact-SQL 構文, 892
ROLLBACK TRIGGER 文
構文, 893
ROLLBACK 文
構文, 890
ROLLUP 演算
GROUP BY 句, 789
WITH ROLLUP 句, 789
ROLLUP オペレーション
GROUPING 関数, 238
ROOT 句
CREATE SERVER 文, 620
ROUND 関数
構文, 335
ROW_NUMBER 関数
構文, 336
rowcount オプション
Transact-SQL SET 文を使用して設定, 929
ROW DELIMITED BY 句
LOAD TABLE 文, 823
UNLOAD 文, 968
RowGenerator
システムテーブル, 1027
ROWID 関数
構文, 337
row limitation 句
SELECT 文, 898
UPDATE 文, 978
row-limitation 句
DELETE 文, 698
ROWS 句

WINDOW 句, 990
R-squared
回帰直線, 321
RTRIM 関数
構文, 338

S

sa_ansi_standard_packages システムプロシージャー
説明, 1042
sa_audit_string システムプロシージャー
構文, 1043
sa_char_terms システムプロシージャー
構文, 1044
sa_check_commit システムプロシージャー
構文, 1045
sa_clean_database システムプロシージャー
構文, 1046
sa_column_stats システムプロシージャー
構文, 1048
sa_conn_activity システムプロシージャー
構文, 1051
sa_conn_compression_info システムプロシージャー
構文, 1052
sa_conn_info システムプロシージャー
構文, 1055
sa_conn_list システムプロシージャー
構文, 1059
sa_conn_options システムプロシージャー
構文, 1060
sa_conn_properties システムプロシージャー
構文, 1061
sa_convert_ml_progress_to_timestamp システムプロシージャー
構文, 1063
sa_convert_timestamp_to_ml_progress システムプロシージャー
構文, 1063
sa_copy_cursor_to_temp_table システムプロシージャー
構文, 1064
sa_db_info システムプロシージャー
構文, 1066
sa_db_list システムプロシージャー
構文, 1067
sa_db_properties システムプロシージャー
構文, 1068
sa_dependent_views システムプロシージャー
構文, 1070
sa_describe_cursor システムプロシージャー
構文, 1071
sa_describe_query システムプロシージャー
構文, 1074
sa_describe_shapefile システムプロシージャー
構文, 1078
sa_diagnostic_auxiliary_catalog テーブル
説明, 1009
sa_diagnostic_blocking テーブル
説明, 1009
sa_diagnostic_cachecontents テーブル
説明, 1011
sa_diagnostic_connection テーブル
説明, 1012
sa_diagnostic_cursor テーブル
説明, 1013
sa_diagnostic_deadlock テーブル
説明, 1014
sa_diagnostic_hostvariable テーブル
説明, 1015
sa_diagnostic_internalvariable テーブル
説明, 1016
sa_diagnostic_query テーブル
説明, 1017
sa_diagnostic_request テーブル
説明, 1019
sa_diagnostic_statement テーブル
説明, 1021
sa_diagnostic_statistics テーブル
説明, 1021
sa_diagnostic_tracing_level テーブル
説明, 1022
sa_disable_auditing_type システムプロシージャー
構文, 1079
sa_disk_free_space システムプロシージャー
構文, 1080
sa_enable_auditing_type システムプロシージャー
構文, 1082
sa_eng_properties システムプロシージャー
構文, 1083
sa_external_library_unload システムプロシージャー
構文, 1084
sa_flush_cache システムプロシージャー
構文, 1085

sa_flush_statistics システムプロシージャー
構文, 1085
sa_get_bits システムプロシージャー
構文, 1086
sa_get_dtt_groupreads システムプロシージャー
構文, 1088
sa_get_dtt システムプロシージャー
構文, 1088
sa_get_histogram システムプロシージャー
構文, 1089
sa_get_request_profile システムプロシージャー
構文, 1091
sa_get_request_times システムプロシージャー
構文, 1092
sa_get_server_messages システムプロシージャー
構文, 1093
sa_get_table_definition システムプロシージャー
構文, 1095
sa_get_user_status システムプロシージャー
構文, 1096
sa_http_header_info システムプロシージャー
構文, 1097
sa_http_php_page_interpreted システムプロシージャー
構文, 1099
sa_http_php_page システムプロシージャー
構文, 1098
sa_http_variable_info システムプロシージャー
構文, 1100
sa_index_density システムプロシージャー
構文, 1101
sa_index_levels システムプロシージャー
構文, 1103
sa_install_feature システムプロシージャー
構文, 1106
sa_java_loaded_classes システムプロシージャー
構文, 1107
sa_list_cursors システムプロシージャー
構文, 1108
sa_load_cost_model システムプロシージャー
構文, 1109
sa_locks システムプロシージャー
構文, 1111
sa_make_object システムプロシージャー
構文, 1115
sa_materialized_view_can_be_immediate システムプロシージャー
構文, 1116
sa_materialized_view_info システムプロシージャー
AvailForOptimization プロパティ, 1119
DataLastModified プロパティ, 1119
OwnerName プロパティ, 1119
Status プロパティ, 1119
ViewName プロパティ, 1119
結果を sa_materialized_view_can_be_immediate と結合, 1124
結果を sa_materialized_view_info と結合, 1124
構文, 1118
例, 1124
sa_migrate_create_fks システムプロシージャー
構文, 1129
sa_migrate_create_remote_fks_list システムプロシージャー
構文, 1131
sa_migrate_create_remote_table_list システムプロシージャー
構文, 1132
sa_migrate_create_tables システムプロシージャー
構文, 1134
sa_migrate_data システムプロシージャー
構文, 1135
sa_migrate_drop_proxy_tables システムプロシージャー
構文, 1136
sa_migrate システムプロシージャー
構文, 1127
sa_mirror_server_status システムプロシージャー
構文, 1137
sa_nchar_terms システムプロシージャー
構文, 1139
sa_performance_diagnostics システムプロシージャー
構文, 1140
sa_performance_statistics システムプロシージャー
構文, 1149
sa_post_login_procedure システムプロシージャー
構文, 1151
sa_procedure_profile_summary システムプロシージャー
構文, 1154
sa_procedure_profile システムプロシージャー
構文, 1152
sa_recompile_views システムプロシージャー
構文, 1157

sa_refresh_materialized_views システムプロシージャー
構文, 1158
sa_refresh_text_indexes システムプロシージャー
構文, 1158
sa_remove_tracing_data システムプロシージャー
構文, 1159
sa_report_deadlocks システムプロシージャー
構文, 1160
sa_reserved_words システムプロシージャー
構文, 1161
sa_reset_identity システムプロシージャー
構文, 1162
sa_rowgenerator システムプロシージャー
構文, 1163
sa_save_trace_data システムプロシージャー
構文, 1165
sa_send_udp システムプロシージャー
構文, 1166
sa_server_messages システムプロシージャー
構文, 1167
sa_server_option システムプロシージャー
構文, 1170
sa_set_http_header システムプロシージャー
構文, 1185
sa_set_http_option システムプロシージャー
構文, 1186
sa_set_soap_header システムプロシージャー
構文, 1190
sa_set_tracing_level システムプロシージャー
構文, 1191
sa_snapshots システムプロシージャー
構文, 1193
sa_split_list システムプロシージャー
構文, 1194
sa_statement_text システムプロシージャー
構文, 1196
sa_table_fragmentation システムプロシージャー
構文, 1197
sa_table_page_usage システムプロシージャー
構文, 1198
sa_table_stats システムプロシージャー
構文, 1199
sa_text_index_stats システムプロシージャー
構文, 1201
sa_text_index_vocab_nchar システムプロシージャー
構文, 1204
sa_text_index_vocab システムプロシージャー
構文, 1202
sa_transactions システムプロシージャー
構文, 1205
sa_unload_cost_model システムプロシージャー
構文, 1206
sa_user_defined_counter_add システムプロシージャー
構文, 1208
sa_user_defined_counter_set システムプロシージャー
構文, 1209
sa_validate システムプロシージャー
構文, 1210
sa_verify_password システムプロシージャー
構文, 1212
SAVE OPTION VALUES 句
ALTER TEXT CONFIGURATION 文, 478
savepoint-name
一般的な SQL 構文要素, 403
SAVEPOINT 文
構文, 895
SAVE TRANSACTION 文
Transact-SQL 構文, 894
SCALE 句
GET DESCRIPTOR 文 [ESQL], 773
SET DESCRIPTOR 文 [ESQL], 907
SCHEDULE 句
CREATE EVENT 文, 543
SCRIPT VERSION 句
CREATE SYNCHRONIZATION
SUBSCRIPTION 文 [Mobile Link], 648
SCROLL カーソル
宣言, 686
SCROLL 句
DECLARE CURSOR 文, 687
FOR 文, 754
search-condition
一般的な SQL 構文要素, 403
SECONDS 関数
構文, 340
SECOND 関数
構文, 339
SecureFeatures プロパティ
sa_server_option での設定, 1183
SECURE 句
ALTER SERVICE 文 [HTTP Web サービス], 441

ALTER SERVICE 文 [SOAP Web サービス], 446
CREATE SERVICE 文 [HTTP Web サービス], 444, 626
CREATE SERVICE 文 [SOAP Web サービス], 451, 633
SELECT
T-SQL の変換, 388
SELECT LIST FOR 句
DESCRIBE 文 [ESQL], 700
SELECT permission 句
GRANT 文, 784
SELECT 句
REVOKE 文, 887
SELECT パーミッション
REVOKE 文, 888
SELECT パーミッション
GRANT 文, 782
SELECT 文
Transact-SQL 構文, 905
構文, 895
ストアドプロシージャーからの選択, 761
データベースオプションの設定, 903
SELECT リスト
カーソルの記述, 700
SELECT リスト句
SELECT 文, 899
self_recursion オプション
Transact-SQL SET 文を使用して設定, 929
SEND AT 句
GRANT CONSOLIDATE 文 [SQL Remote], 776
GRANT REMOTE 文 [SQL Remote], 780
PUBLISH, 777
説明, 776
SEND EVERY 句
GRANT CONSOLIDATE 文 [SQL Remote], 776
GRANT REMOTE 文 [SQL Remote], 780
説明, 776
SENSITIVE 句
DECLARE CURSOR 文, 687
FOR 文, 754
SERIALIZABLE 句
REFRESH MATERIALIZED VIEW 文, 867
SERIALIZABLE テーブルヒント
FROM 句, 764
Server
HTTP_RESPONSE_HEADER 関数, 251
ServerIdle システムイベント
例, 843
server-option 句
ALTER MIRROR SERVER 文, 432
CREATE MIRROR SERVER 文, 581
SessionTimeout オプション
sa_set_http_option システムプロシージャー, 1186
SET_BITS 関数
構文, 343
SET_BIT 関数
構文, 341
SET CONNECTION 文
Embedded SQL 構文, 906
Interactive SQL 構文, 906
SET DESCRIPTOR 文
Embedded SQL 構文, 907
SET HIDDEN 句
ALTER EVENT 文, 419
ALTER FUNCTION 文, 421
ALTER MATERIALIZED VIEW 文, 427
ALTER PROCEDURE 文, 433
ALTER VIEW 文, 485
SET MIRROR OPTION 文
構文, 908
SET OPTION 文
Interactive SQL 構文, 916
Transact-SQL 構文, 929
構文, 913
SET PARTNER FAILOVER 句
ALTER DATABASE 文, 410
SET PERMANENT 文
Interactive SQL 構文, 916
SET REMOTE OPTION 文
ftp オプション, 922
http オプション, 922
smtp オプション, 924
SQL Remote 構文, 919
共通オプション, 921
ファイルオプション, 921
SET SCRIPT VERSION 句
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
START SYNCHRONIZATION SCHEMA CHANGE 文, 944
SET SESSION AUTHORIZATION 文
構文, 932
SET SQLCA 文
Embedded SQL 構文, 926

SET TEMPORARY OPTION 文
Interactive SQL 構文, 916
構文, 913
SETUSER 文
構文, 932
SET 句
CREATE FUNCTION 文 [Web クライアント], 559
CREATE PROCEDURE 文 [Web クライアント], 595
UPDATE (位置付け) 文, 971
UPDATE 文, 978
UPDATE 文 [SQL Remote], 973
リダイレクション URI, 595
SET 文
Interactive SQL 構文, 916
Transact-SQL 構文, 929
構文, 927
SHARE BY ALL 句
CREATE TABLE 文, 654
SIGNAL 文
構文, 934
SIGN 関数
構文, 344
SIMILAR TO 探索条件
REGEXP と LIKE との比較, 40
構文, 48
正規表現, 18
データベース照合と一致, 49
部分文字クラスの一致, 48
SIMILAR 関数
構文, 344
SIN 関数
構文, 345
SIZE 句
ATTACH TRACING 文, 487
SKIP 句
LOAD TABLE 文, 824
MERGE 文, 838
SMALLDATETIME データ型
構文, 116
日付と時刻をデータベースに送信する, 106
SMALLINT データ型
構文, 101
日付と時刻の比較, 130
SMALLMONEY データ型
構文, 103
SMTP
拡張システムプロシージャー, 1030
電子メールセッションの開始, 1247
電子メールセッションの終了, 1249
リターンコード, 1031
SMTP 句
GRANT CONSOLIDATE 文 [SQL Remote], 776
SNAPSHOT 句
REFRESH MATERIALIZED VIEW 文, 867
SOAP
CREATE SERVICE 文 [SOAP Web サービス], 447, 629
SOAP_HEADER 関数
構文, 346
SOAPHEADER 句
CREATE FUNCTION 文 [Web クライアント], 557
CREATE PROCEDURE 文 [Web クライアント], 597
SOAP 関数
アルファベット順リスト, 147
SOAP サービス
データ型指定, 448, 630
SOAP システムプロシージャー
アルファベット順リスト, 1030
SOAP ヘッダー
設定, 1190
SOME 探索条件
構文, 38
SORTKEY 関数
構文, 347
照合の適合化, 347
SOUNDEX 関数
構文, 350
SP
文インジケーター, 406
sp_addgroup システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_addlogin システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_addmessage システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
説明, 578
sp_addtype システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033

sp_adduser システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_changegroup システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_column_privileges カタログプロシージャー
説明, 1034
sp_columns カタログプロシージャー
説明, 1034
sp_dropgroup システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_droplogin システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_dropmessage システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_droptype システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_dropuser システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_fkeys カタログプロシージャー
説明, 1034
sp_forward_to_remote_server システムプロシージャー
構文, 1212
sp_get_last_synchronize_result システムプロシージャー
構文, 1214
sp_getmessage システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_helptext システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_login_environment システムプロシージャー
構文, 1217
sp_password システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
sp_pkeys カタログプロシージャー
説明, 1034
sp_remote_columns システムプロシージャー
構文, 1218
sp_remote_exported_keys システムプロシージャー
構文, 1219
sp_remote_imported_keys システムプロシージャー
構文, 1221
sp_remote_primary_keys システムプロシージャー
構文, 1223
sp_remote_tables システムプロシージャー
構文, 1224
sp_servercaps システムプロシージャー
構文, 1226
sp_special_columns カタログプロシージャー
説明, 1034
sp_sproc_columns カタログプロシージャー
説明, 1034
sp_statistics カタログプロシージャー
説明, 1034
sp_stored_procedures カタログプロシージャー
説明, 1034
sp_tables カタログプロシージャー
説明, 1034
sp_tsql_environment システムプロシージャー
構文, 1227
SPACE 関数
構文, 351
special-value
一般的な SQL 構文要素, 404
Specification プロパティ
DB_EXTENDED_PROPERTY 関数, 203
SQL
SQL Anywhere サーバーの文のアルファベット順リスト, 406
SQL/1999
SQL 準拠のテスト, 352
SQL/2008
SQL 準拠のテスト, 352
SQL/2003
SQL 準拠のテスト, 352
SQLCA
INCLUDE 文, 794
設定, 926
SQLCA 句
INCLUDE 文, 794
SQLCODE
特別値, 70
SQLDA
DESCRIBE 文 [ESQL], 700
EXECUTE 文, 742
INCLUDE 文, 794

UPDATE (位置付け) 文, 971
カーソルを使用したローの挿入, 860
情報の取得, 773
設定, 907
メモリの割り付け, 406
割り付けの解除, 684
SQLDA 句
INCLUDE 文, 794
SQLDescribeCol
CHAR データ型の説明, 85
NCHAR データ型の説明, 88
NVARCHAR データ型の説明, 89
VARCHAR データ型の説明, 91
SQLDIALECT 関数
構文, 352
SQLERROR 句
WHENEVER 文 [ESQL], 987
SQL Flagger
SQLFLAGGER 関数, 352
コア以外の拡張機能に対する SQL 文のテスト, 1042
SQLFLAGGER 関数
構文, 352
SQL Remote
Remote オプションの設定, 919
SYSARTICLECOL システムビュー, 1255
SYSARTICLE システムビュー, 1254
SYSPUBLICATIONS 統合ビュー, 1348
SYSPUBLICATION システムビュー, 1291
SYSREMOTEOPTIONS 統合ビュー, 1349
SYSREMOTEOPTIONTYPE システムビュー, 1294
SYSREMOTEOPTION システムビュー, 1293
SYSREMOTETYPES 統合ビュー, 1349
SYSREMOTETYPE システムビュー, 1294
SYSREMOTEUSERS 統合ビュー, 1350
SYSREMOTEUSER システムビュー, 1295
アーティクル SYSARTICLE, 1254
アーティクル SYSARTICLECOL, 1255
サブスクリプションの作成, 645
SQL Remote システムビュー
SYSARTICLECOL, 1255
SYSPUBLICATIONS 統合ビュー, 1348
SYSPUBLICATION システムビュー, 1291
SYSREMOTEOPTION, 1293
SYSREMOTEOPTIONS 統合ビュー, 1349
SYSREMOTEOPTIONTYPE, 1294
SYSREMOTETYPES 統合ビュー, 1349
SYSREMOTEUSER, 1295
SYSREMOTEUSERS 統合ビュー, 1350
アーティクルシステムビュー, 1254
SQL SECURITY 句
CREATE FUNCTION 文 [外部プロシージャー], 564
CREATE FUNCTION 文 [外部呼び出し], 550
CREATE PROCEDURE 文, 605
CREATE PROCEDURE 文 [外部呼び出し], 587
SQL Server
sa_migrate システムプロシージャーを使用して SQL Anywhere にマイグレート, 1128
SQLSetConnectAttr
MESSAGE TO CLIENT と使用, 844
SQLSTATE
ISO/ANSI 標準への適合, 71
特別値, 71
SQLWARNING 句
WHENEVER 文 [ESQL], 987
SQL から Java のデータ型変換
説明, 137
SQL 関数
HTTP, 147
NULL 引数を指定した場合は NULL を返す, 139
SOAP, 147
イメージ, 151
概要, 139
関数のタイプ, 139
システム, 150
集合, 139
数値, 146
その他, 146
テキスト, 151
データ型変換, 141
日付と時刻, 142
ビット配列, 141
文字列, 148
ユーザー定義, 144
ランキング, 141
SQL 規格
準拠のテスト, 352
SQL キーワード
sa_reserved_words システムプロシージャー, 1161
予約語のリスト, 1
SQL 句の順序
構文の表記規則, 404

SQL 言語要素
説明, 1
SQL 構文
3 値的論理, 59
ALL 探索条件, 37
ANY 探索条件, 38
BETWEEN 探索条件, 39
CASE 式, 16
CONTAINS 探索条件, 51
CURRENT_TIMESTAMP 特別値, 64
CURRENT_USER 特別値, 65
CURRENT DATABASE 特別値, 62
CURRENT DATE 特別値, 62
CURRENT PUBLISHER 特別値, 62
CURRENT REMOTE USER 特別値, 63
CURRENT TIMESTAMP 特別値, 64
CURRENT TIME 特別値, 63
CURRENT USER 特別値, 65
CURRENT UTC TIMESTAMP 特別値, 66
EXISTS 探索条件, 57
IF 式, 15
IN 探索条件, 50
IS NOT NULL 探索条件, 57
IS NULL 探索条件, 57
IS TRUE または FALSE 探索条件, 58
LAST USER 特別値, 67
LIKE 探索条件, 42
NULL 値, 67
REGEXP 探索条件, 47
SIMILAR TO 探索条件, 48
SOME 探索条件, 38
SQL Anywhere サーバーの文のアルファベット順リスト, 406
SQLCODE 特別値, 70
SQLSTATE 特別値, 71
TIMESTAMP 特別値, 73
Transact-SQL 式の互換性, 33
USER 特別値, 74
UTC TIMESTAMP 特別値, 75
演算子, 8
演算子の優先度, 12
カラム名, 14
関数, 139
キーワード, 1
コメント, 84
サブクエリ, 15
算術演算子, 10
式, 12
式内の定数, 14
識別子, 4
システムプロシージャーのアルファベット順リスト, 1035
述部, 34
接続レベル変数, 78
探索条件, 34
探索条件内のサブクエリ, 36
定数, 6
特別値, 61
比較演算子, 8
ビット処理演算子, 11
変数, 76
マニュアルの表記規則, 403
文字列, 5
文字列演算子, 11
論理演算子, 9
ローカル変数, 77
SQL スクリプトファイル
Interactive SQL のパラメーター, 854
SQL 文の読み込み, 863
SQL での句の順序
構文の表記規則, 404
SQL 文
Java クラスのインストール, 809
SQL Anywhere サーバーの文のアルファベット順リスト, 406
マニュアルの表記規則, 403
リモートサーバーへの送信, 757
SQL 変数
DROP VARIABLE 文を使用して削除, 736
値の設定, 927
作成, 679
宣言, 693
SQL 文字列の区切り
説明, 4
SQRT 関数
構文, 353
SRID
空間参照系を作成する場合の SRID の選択, 636
ST_GEOMETRY_COLUMNS
統合ビュー, 1335
st_geometry_dump システムプロシージャー
構文, 1228
st_geometry_load_shapefile システムプロシージャー
構文, 1235

ST_SPATIAL_REFERENCE_SYSTEMS
統合ビュー, 1336
ST_UNITS_OF_MEASURE
統合ビュー, 1340
START AT 句
DELETE 文, 698
SELECT 文, 898
UPDATE 文, 978
START DATABASE 文
構文, 935
START DATE 句
ALTER EVENT 文, 419
CREATE EVENT 文, 544
START EXTERNAL ENVIRONMENT 文
構文, 939
START JAVA 文
構文, 940
START LOGGING 文
Interactive SQL 構文, 940
START SERVER 文
Interactive SQL 構文, 938
START SUBSCRIPTION 文
SQL Remote 構文, 941
START SYNCHRONIZATION DELETE 文
Mobile Link 構文, 943
START SYNCHRONIZATION SCHEMA CHANGE 文
Mobile Link 構文, 944
START TIME 句
ALTER EVENT 文, 419
CREATE EVENT 文, 543
START WITH 句
CREATE SEQUENCE 文, 617
state_file オプション
ALTER MIRROR SERVER 文, 432
CREATE MIRROR SERVER 文, 582
statement label
一般的な SQL 構文要素, 404
statement-list
一般的な SQL 構文要素, 404
StatisticsCleaner プロパティ
sa_server_option での設定, 1184
STATISTICS 句
LOAD TABLE 文, 825
Status プロパティ
sa_materialized_view_info システムプロシージャー, 1119
STDDEV_POP 関数
構文, 354
STDDEV_SAMP 関数
構文, 356
STDDEV 関数
構文, 354
STOP DATABASE 句
STOP DATABASE 文, 946
STOP DATABASE 文
構文, 946
STOP EXTERNAL ENVIRONMENT 文
構文, 947
STOP JAVA 文
構文, 948
STOPLIST 句
ALTER TEXT CONFIGURATION 文, 477
STOP LOGGING 文
Interactive SQL 構文, 949
STOP SERVER 文
構文, 949
STOP SUBSCRIPTION 文
SQL Remote 構文, 950
STOP SYNCHRONIZATION DELETE 文
Mobile Link 構文, 951
STOP SYNCHRONIZATION SCHEMA CHANGE 文
Mobile Link 構文, 957
STOP 句
WHENEVER 文 [ESQL], 987
STORAGE FORMAT 句
ALTER SPATIAL REFERENCE SYSTEM 文, 455
CREATE SPATIAL REFERENCE SYSTEM 文, 640
string_rtruncation オプション
Transact-SQL SET 文を使用して設定, 929
string-expression
一般的な SQL 構文要素, 404
STRING 関数
構文, 359
STRIP 句
LOAD TABLE 文, 824
STRTOUUID 関数
構文, 359
STR 関数
構文, 358
STUFF 関数
構文, 361
su

ユーザーの設定, 932
SUBDIRS 句
CREATE SERVER 文, 620
SUBSCRIBE BY 句
ALTER PUBLICATION 文, 435
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 611
UPDATE 文, 976
UPDATE 文 [SQL Remote], 973
SUBSTRING 関数
構文, 361
SUBSTR 関数
構文, 361
SUM 関数
構文, 363
SUSER_ID 関数
構文, 365
SUSER_NAME 関数
構文, 365
SWITCHOFFSET 関数
構文, 366
synchronization_mode オプション
SET MIRROR OPTION 文, 909
SYNCHRONIZE SUBSCRIPTION 文
SQL Remote 構文, 958
SYNCHRONIZE 文
Mobile Link 構文, 952
SYS
システムテーブル, 993
デフォルトのシステムビュー, 1253
SYSARTICLE
システムビュー, 1254
SYSARTICLECOL
システムビュー, 1255
SYSARTICLECOLS
統合ビュー, 1340
SYSARTICLES
統合ビュー, 1341
SYSCAPABILITIES
統合ビュー, 1341
SYSCAPABILITY
システムビュー, 1255
SYSCAPABILITYNAME
システムビュー, 1256
SYSCATALOG
統合ビュー, 1342
SYSCHECK
システムビュー, 1256
SYSCOLAUTH
統合ビュー, 1342
SYSCOLLATION
互換ビュー (旧式), 1356
SYSCOLLATIONMAPPINGS
互換ビュー (旧式), 1357
SYSCOLPERM
システムビュー, 1257
SYSCOLSTAT
システムビュー, 1258
SYSCOLSTATS
統合ビュー, 1343
SYSCOLUMN
互換ビュー (旧式), 1357
SYSCOLUMNS
統合ビュー, 1343
SYSCONSTRAINT
システムビュー, 1259
SYSDATETIMEOFFSET 関数
構文, 367
SYSDBFILE
システムビュー, 1259
SYSDBSPACE
システムビュー, 1260
SYSDBSPACEPERM
システムビュー, 1261
SYSDEPENDENCY
システムビュー, 1262
SYSDOMAIN
システムビュー, 1262
SYSEVENT
システムビュー, 1263
SYSEVENTTYPE
システムビュー, 1264
SYSEXTERNENV
システムビュー, 1265
SYSEXTERNENVOBJECT
システムビュー, 1266
SYSEXTERNLOGIN
システムビュー, 1267
SYSFILE
互換ビュー (旧式), 1358
SYSFKCOL
互換ビュー (旧式), 1358
SYSFKEY
システムビュー, 1268
SYSFOREIGNKEY
互換ビュー (旧式), 1359

SYSFOREIGNKEYS
統合ビュー, 1344
SYSGROUP
システムビュー, 1269
SYSGROUPS
統合ビュー, 1345
SYSHISTORY
システムビュー, 1269
SYSIDX
システムビュー, 1273
SYSIDXCOL
システムビュー, 1275
SYSINDEX
互換ビュー (旧式), 1359
SYSINDEXES
統合ビュー, 1345
SYSINFO
互換ビュー (旧式), 1360
SYSIXCOL
互換ビュー (旧式), 1361
SYSJAR
システムビュー, 1276
SYSJARCOMPONENT
システムビュー, 1277
SYSJAVACLASS
システムビュー, 1277
SYSLOGINMAP
システムビュー, 1278
SYSLOGINPOLICY
システムビュー, 1279
SYSLOGINPOLICYOPTION
システムビュー, 1280
SYSMIRROROPTION
システムビュー, 1280
SYSMIRRORSERVER
システムビュー, 1280
SYSMIRRORSERVEROPTION
システムビュー, 1281
SYSMVOPTION
システムビュー, 1282
SYSMVOPTIONNAME
システムビュー, 1283
SYSOBJECT
システムビュー, 1283
SYSOPTION
システムビュー, 1285
SYSOPTIONS
統合ビュー, 1346
SYSOPTSTAT
システムビュー, 1285
SYSPHYSIDX
システムビュー, 1285
SYSPROCAUTH
統合ビュー, 1346
SYSPROCEDURE
システムビュー, 1286
SYSPROCPARM
システムビュー, 1288
SYSPROCPARMS
統合ビュー, 1347
SYSPROCPERM
システムビュー, 1290
SYSPROCS
統合ビュー, 1347
SYSPROXYTAB
システムビュー, 1291
SYSPUBLICATION
システムビュー, 1291
SYSPUBLICATIONS
統合ビュー, 1348
SYSREMARK
システムビュー, 1293
SYSREMOTEOPTION
システムビュー, 1293
SYSREMOTEOPTION2
統合ビュー, 1348
SYSREMOTEOPTIONS
統合ビュー, 1349
SYSREMOTEOPTIONTYPE
システムビュー, 1294
SYSREMOTETYPE
システムビュー, 1294
SYSREMOTETYPES
統合ビュー, 1349
SYSREMOTEUSER
システムビュー, 1295
SYSREMOTEUSERS
統合ビュー, 1350
SYSSCHEDULE
システムビュー, 1297
SYSSEQUENCE
システムビュー, 1299
SYSSEQUENCEPERM
システムビュー, 1300
SYSSERVER
システムビュー, 1301

SYSSOURCE
システムビュー, 1301
SYSSPATIALREFERENCESYSTEM
システムビュー, 1302
SYSSQLSERVERTYPE
システムビュー, 1306
SYSSUBSCRIPTION
システムビュー, 1307
SYSSUBSCRIPTIONS
統合ビュー, 1350
SYSSYNC
システムビュー, 1308
SYSSYNC2
統合ビュー, 1351
SYSSYNCPROFILE
システムビュー, 1309
SYSSYNCPUBLICATIONDEFAULTS
統合ビュー, 1351
SYSSYNCS
統合ビュー, 1352
SYSSYNCSCRIPT
システムビュー, 1310
SYSSYNCSCRIPTS
統合ビュー, 1352
SYSSYNCSUBSCRIPTIONS
統合ビュー, 1353
SYSSYNCUSERS
統合ビュー, 1353
SYSTAB
互換ビューの情報, 993
システムビュー, 1311
SYSTABAUTH
統合ビュー, 1354
SYSTABCOL
システムビュー, 1315
SYSTABLE
互換ビュー (旧式), 1361
SYSTABLEPERM
システムビュー, 1318
SYSTEM 文
dbisqlc でサポートされない, 959
Interactive SQL 構文, 959
SYSTEXTCONFIG
システムビュー, 1320
SYSTEXTIDX
システムビュー, 1322
SYSTEXTIDXTAB
システムビュー, 1324
SYSTRIGGER
システムビュー, 1324
SYSTRIGGERS
統合ビュー, 1354
SYSTYPEMAP
システムビュー, 1327
SYSUNITOFMEASURE
システムビュー, 1327
SYSUSER
システムビュー, 1328
SYSUSERAUTH
互換ビュー (旧式), 1362
SYSUSERAUTHORITY
システムビュー, 1329
SYSUSERLIST
互換ビュー (旧式), 1363
SYSUSERMESSAGE
システムビュー, 1329
SYSUSEROPTIONS
統合ビュー, 1355
SYSUSERPERM
互換ビュー (旧式), 1363
SYSUSERPERMS
互換ビュー (旧式), 1364
SYSUSERTYPE
システムビュー, 1330
SYSVIEW
システムビュー, 1331
SYSVIEWS
統合ビュー, 1356
統合ビューの情報, 993
SYSWEBSERVICE
システムビュー, 1333

T

TABLE.COLUMN 句
DESCRIBE 文 [ESQL], 700
table-list
一般的な SQL 構文要素, 404
table-name
一般的な SQL 構文要素, 404
TABLE 句
ALTER PUBLICATION 文, 435
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
DESCRIBE 文, 704
TRUNCATE 文, 961

VALIDATE 文, 983
TAN 関数
構文, 367
TempFreePercent イベント条件
説明, 222
TempFreeSpace イベント条件
説明, 222
temporary DB 領域
調整, 409
TEMPORARY キーワード
CREATE FUNCTION 文 [ユーザー定義], 563
TEMPORARY 句
ALTER DBSPACE 文, 414
CREATE PROCEDURE 文, 602
TempSize イベント条件
説明, 222
TERM BREAKER EXTERNAL NAME 句
ALTER TEXT CONFIGURATION 文, 476
TERM BREAKER 句
ALTER TEXT CONFIGURATION 文, 477
TEXTPTR 関数
構文, 368
textsize オプション
Transact-SQL SET 文を使用して設定, 929
TEXT データ型
構文, 90
THEN
IF 式, 15
THEN 句
IF 文, 791
TIMESTAMP WITH TIME ZONE データ型
構文, 120
日付と時刻の比較, 130
日付と時刻をデータベースに送信する, 106
TIMESTAMP 句
CREATE TABLE 文, 651
TIMESTAMP デフォルト
CREATE TABLE 文, 465, 656
TIMESTAMP データ型
構文, 118
日付と時刻の比較, 130
日付と時刻のデータベースに送信する, 106
TIMESTAMP 特別値
構文, 73
TIME 句
WAITFOR 文, 985
TIME データ型
構文, 117
日付と時刻の比較, 130
日付と時刻のデータベースに送信する, 106
TINYINT データ型
構文, 102
TO_CHAR 関数
構文, 369
TO_NCHAR 関数
構文, 370
TODATETIMEOFFSET 関数
構文, 371
TODAY 関数
構文, 372
TOLERANCE 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 639
TOP 句
DELETE 文, 698
SELECT 文, 898
UPDATE 文, 978
TO 句
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
CONNECT 文, 516
CREATE EXTERNLOGIN 文, 548
CREATE SUBSCRIPTION 文 [SQL Remote], 645
CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 648
DROP EXTERNLOGIN 文, 714
DROP SYNCHRONIZATION SUBSCRIPTION 文, 730
GRANT CONSOLIDATE 文 [SQL Remote], 776
GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote], 778
MESSAGE 文, 841
PASSTHROUGH 文 [SQL Remote], 855
START SUBSCRIPTION 文 [SQL Remote], 941
STOP SUBSCRIPTION 文 [SQL Remote], 950
SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote], 958
UNLOAD 文, 966
TRACEBACK 関数
構文, 372
TRACED_PLAN 関数
構文, 373
TRANSACTION LOG ONLY 句
BACKUP 文, 490

TRANSACTION LOG RENAME [MATCH] 句
BACKUP 文, 490
TRANSACTION LOG TRUNCATE 句
BACKUP 文, 490
TRANSACTION LOG 句
CREATE DATABASE 文, 526
Transact-SQL
ANSI への書き換え, 332
CASE 文 [T-SQL], 507
CONTINUE Transact-SQL 構文, 988
CREATE FUNCTION 文, 567
CREATE MESSAGE 文, 578
CREATE PROCEDURE 文, 599
CREATE SCHEMA SQL 構文, 615
CREATE TABLE SQL 構文, 663
CREATE TRIGGER SQL 構文, 676
DECLARE CURSOR SQL 構文, 690
EXECUTE SQL 構文, 744
GOTO SQL 構文, 775
IF SQL 構文, 793
PRINT SQL 構文, 859
quoted_identifier オプション, 34
READTEXT SQL 構文, 866
SELECT 文, 905
SELECT 文の変換, 388
SET OPTION SQL 構文, 929
SET SQL 構文, 929
SQL Anywhere サーバーの文のアルファベット順リスト, 406
SQL 式の互換性, 33
WHILE SQL 構文, 988
WRITETEXT SQL 構文, 991
外部ジョイン演算子, 12
カタログプロシージャー, 1034
時刻の互換性, 132
システム関数, 150
システムプロシージャー, 1033
システムプロシージャーとカタログプロシージャー, 1033
ストアドプロシージャーの変換, 388
通貨データ型, 103
定数, 33
ドメイン, 126
比較演算子, 8
日付と時刻の互換性, 132
ビット処理演算子, 11
文インジケーター, 406
文字列, 33
ユーザー定義データ型, 126
ローカル変数, 77
TRANSACTSQL 関数
構文, 373
Transact-SQL との互換性
グローバル変数, 79
ビュー, 1365
Transact-SQL の文字列から日付／時刻への変換
説明, 132
Transact-SQL 文
BEGIN TRANSACTION 構文, 500
ROLLBACK TRANSACTION 構文, 892
SAVE TRANSACTION 構文, 894
Transfer-Encoding
HTTP_RESPONSE_HEADER 関数, 251
TRANSFORM DEFINITION 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
TRANSLOG 句
ALTER DBSPACE 文, 414
TREAT 関数
構文, 374
TRIGGER EVENT 文
構文, 960
TRIM 関数
構文, 375
TRUE 条件
3 値的論理, 59
IS TRUE 探索条件, 58
TRUNCATE MATERIALIZED VIEW 文
構文, 961
TRUNCATE TABLE 文
構文, 961
TRUNCATE TEXT INDEX 文
構文, 963
TRUNCATE 関数
構文, 376
TRUNCATE 文
構文, 961
TRUNCNUM 関数
構文, 376
TSEQUAL 関数
構文, 377
[T-SQL]
文インジケーター, 406
TYPE PLANAR 句

ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
TYPE ROUND EARTH 句
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
TYPE 句
ALTER SERVICE 文 [HTTP Web サービス], 441
ALTER SERVICE 文 [SOAP Web サービス], 446
ALTER SPATIAL REFERENCE SYSTEM 文, 454
ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 459
ALTER SYNCHRONIZATION USER 文 [Mobile Link], 461
CREATE EVENT 文, 541
CREATE FUNCTION 文 [Web クライアント], 557
CREATE PROCEDURE 文 [Web クライアント], 593
CREATE SERVICE 文 [HTTP Web サービス], 441, 623
CREATE SERVICE 文 [SOAP Web サービス], 447, 629
CREATE SPATIAL REFERENCE SYSTEM 文, 637
CREATE SPATIAL UNIT OF MEASURE 文, 642
CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 648
CREATE SYNCHRONIZATION USER, 650
GET DESCRIPTOR 文 [ESQL], 773
GRANT CONSOLIDATE 文 [SQL Remote], 776
GRANT REMOTE 文 [SQL Remote], 780
MESSAGE 文, 841
SET DESCRIPTOR 文 [ESQL], 907

U

UCASE 関数
構文, 378
UDF
ユーザー定義関数、定義済み, 144
UDP パケット
送信, 1166
Ultra Light
NULL 引数を指定した場合は NULL を返す, 139
UNBOUNDED キーワード
WINDOW 句の FOLLOWING 句, 990
WINDOW 句の PRECEDING 句, 990
UNCONDITIONALLY 句
STOP DATABASE 文, 946
STOP SERVER 文, 949
Unicode
UNICODE 関数, 379
UNISTR 関数, 380
エスケープシーケンス, 380
UNICODE 関数
構文, 379
Unicode データ
格納, 85
Unicode データ型
説明, 85
UNION 文
構文, 964
データベースオプションの設定, 964
UNIQUEIDENTIFIERSTR データ型
構文, 91
UNIQUEIDENTIFIER データ型
構文, 123
UNIQUE 句
ALTER TABLE 文, 467
CREATE INDEX 文, 568
CREATE TABLE 文, 661
DECLARE CURSOR 文, 686
UNISTR 関数
構文, 380
UNIX
文字列の圧縮, 173
文字列の解凍, 210
UNIX で文字列を圧縮する
COMPRESS 関数, 173
UNKNOWN 条件
IS UNKNOWN 探索条件, 58
UNLIMITED 句
CREATE LOGIN POLICY 文, 573
UNLOAD TABLE 文
構文, 966
UNLOAD 文
構文, 966
UNSIGNED BIGINT データ型
構文, 94

UNSIGNED INTEGER データ型
構文, 99
UNSIGNED SMALLINT データ型
構文, 101
UNSIGNED TINYINT データ型
構文, 102
unzip ユーティリティ
DECOMPRESS 関数, 210
UPDATE permission 句
GRANT 文, 784
UPDATE SET 句
MERGE 文, 838
UPDATE 句
REVOKE 文, 887
UPDATE 句
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
CREATE TRIGGER 文, 671
INSTALL EXTERNAL OBJECT 文, 807
UPDATE パーミッション
REVOKE 文, 888
UPDATE パーミッション
GRANT 文, 782
UPDATE 文
SQL Remote 構文, 973
(位置付け) 文の構文, 971
構文, 976
データベースオプションの設定, 979
UPDLOCK テーブルヒント
FROM 句, 764
UPPER 関数
構文, 381
URL 句
ALTER SERVICE 文 [HTTP Web サービス], 441
CREATE FUNCTION 文 [Web クライアント], 556
CREATE PROCEDURE 文 [Web クライアント], 593
CREATE SERVICE 文 [HTTP Web サービス], 442, 624
URL リダイレクション
CREATE PROCEDURE 文 [Web クライアント], 595
USAGE ON SEQUENCE 句
REVOKE 文, 887
USAGE ON SEQUENCE 句
GRANT 文, 782
USE DEFAULT 句
CREATE TABLE 文, 651
USER
CONNECT 文, 516
user_estimates オプション
DELETE 文に設定, 699
EXCEPT 文に設定, 738
INSERT 文に設定, 804
INTERSECT 文に設定, 811
MERGE 文での上書き, 839
SELECT 文での上書き, 903
UNION 文に設定, 964
UPDATE 文に設定, 979
USER_ID 関数
構文, 381
USER_NAME 関数
構文, 382
User-Agent
HTTP_HEADER 関数, 249
HTTP_RESPONSE_HEADER 関数, 251
userid
一般的な SQL 構文要素, 404
USER TYPES 句
DESCRIBE 文, 701
USER 句
ALTER SERVICE 文 [HTTP Web サービス], 441
ALTER SERVICE 文 [SOAP Web サービス], 446
CREATE SERVICE 文 [HTTP Web サービス], 445, 626
CREATE SERVICE 文 [SOAP Web サービス], 451, 633
CREATE TABLE 文, 651
USER 特別値
構文, 74
USING AUTO PARENT 句
ALTER MIRROR SERVER 文, 431
CREATE MIRROR SERVER 文, 581
USING CLIENT FILE 句
LOAD TABLE 文, 816
USING COLUMN 句
LOAD TABLE 文, 817
USING DESCRIPTOR 句
EXPLAIN 文 [ESQL], 747
GET DATA 文, 771
GET OPTION 文 [ESQL], 774
OPEN 文, 845
PUT 文 [ESQL], 860
UPDATE (位置付け) 文, 971

USING FILE 句
LOAD TABLE 文, 816
USING VALUE 句
LOAD TABLE 文, 817
USING 句
ALTER PUBLICATION 文, 435
ALTER SERVER 文, 439
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
CREATE SERVER 文, 619
DECLARE CURSOR 文, 687
EXECUTE 文, 743
FOR 文, 753
INPUT 文, 799
MERGE 文, 836
OUTPUT 文, 851
UTC TIMESTAMP 句
CREATE TABLE 文, 651
UTC TIMESTAMP デフォルト
CREATE TABLE 文, 465, 656
UTC TIMESTAMP 特別値
構文, 75
UTF-16 エンコード
CSCONVERT 関数, 189
LOAD TABLE 文, 818, 820
UNLOAD 文, 967
UUID
NEWID 関数の SQL 構文, 286
STRTOUUID 関数の SQL 構文, 359
UNIQUEIDENTIFIER データ型, 123
UUIDTOSTR 関数の SQL 構文, 383
UUIDTOSTR 関数
構文, 383

V

VALIDATE authority 句
GRANT 文, 784
VALIDATE CHECKSUM 文
構文, 983
VALIDATE DATABASE 文
構文, 983
VALIDATE INDEX 文
構文, 983
VALIDATE MATERIALIZED VIEW 文
構文, 983
VALIDATE TABLE 文
構文, 983
VALIDATE TEXT INDEX 文
構文, 983
VALIDATE 句
REVOKE 文, 887
VALIDATE 権限
REVOKE 文, 888
VALIDATE 権限
GRANT 文, 782
VALIDATE 文
構文, 983
VALUES 句
INSERT 文, 802
VALUE 句
GET DESCRIPTOR 文 [ESQL], 773
SET DESCRIPTOR 文 [ESQL], 907
VAR_POP 関数
構文, 384
VAR_SAMP 関数
構文, 386
VARBINARY データ型
構文, 124
VARBIT データ型
構文, 105
VARCHAR データ型
VARCHAR カラムに DESCRIBE を使用, 91
構文, 91
バイト長のセマンティック, 91
文字長のセマンティック, 91
VAREXISTS 関数
構文, 387
variable-name
一般的な SQL 構文要素, 404
VARIANCE 関数
構文, 388
VERBOSE 句
OUTPUT 文, 851
VERIFY 句
UPDATE 文, 976
UPDATE 文 [SQL Remote], 973
ViewName プロパティ
sa_materialized_view_info システムプロシージャー, 1119
VIRTUAL 句
CREATE INDEX 文, 568
VM
START JAVA 文, 940
STOP JAVA 文, 948

W

WAIT AFTER END 句
BACKUP 文, 490
WAIT BEFORE START 句
BACKUP 文, 490
WAITFOR 文
構文, 985
Watcom SQL
DECLARE 文, 693
WATCOMSQL 関数
構文, 388
Watcom SQL 文
Transact SQL への書き換え, 373
WebClientLogFile プロパティ
sa_server_option での設定, 1184
WebClientLogging プロパティ
sa_server_option での設定, 1184
Web クライアント
CREATE FUNCTION 文 [Web サービス] を使用して Web クライアント関数を作成, 555
Web サービス
ALTER SERVICE 文 [SOAP Web サービス] を使用して SOAP サービスを変更, 446
COMMENT 文を使用してコメントを追加, 512
CREATE SERVICE 文 [HTTP Web サービス] を使用して作成, 622
CREATE SERVICE 文 [SOAP Web サービス] を使用して作成, 628
HTML_DECODE 関数, 244
HTML_ENCODE 関数, 245
HTTP_BODY 関数, 246
HTTP_DECODE 関数, 247
HTTP_ENCODE 関数, 248
HTTP_HEADER 関数, 249
HTTP_RESPONSE_HEADER 関数, 251
HTTP_VARIABLE 関数, 253
NEXT_HTTP_HEADER 関数, 290
NEXT_HTTP_RESPONSE_HEADER 関数, 291
NEXT_HTTP_VARIABLE 関数, 292
NEXT_SOAP_HEADER 関数, 293
sa_http_header_info システムプロシージャー, 1097
sa_http_php_page_interpreted システムプロシージャー, 1099
sa_http_php_page システムプロシージャー, 1098
sa_http_variable_info システムプロシージャー, 1100
sa_set_http_header_info システムプロシージャー, 1185
sa_set_http_option システムプロシージャー, 1186
sa_set_soap_header システムプロシージャー, 1190
SOAP_HEADER 関数, 346
Web サービス関連システムプロシージャーのリスト, 1030
アルファベット順の関数リスト, 147
アルファベット順のシステムプロシージャー, 1030
システムビュー, 1333
Web サービスクライアントログファイル
名前の設定, 1184
WEEKS 関数
構文, 389
WHEN
CASE 式, 16
WHENEVER 文
Embedded SQL 構文, 987
WHEN MATCHED 句
MERGE 文, 837
WHEN NOT MATCHED 句
MERGE 文, 837
WHEN 句
CASE 文, 505
CASE 文 [T-SQL], 507
CREATE TRIGGER 文, 672
WHERE CURRENT OF 句
DELETE (位置付け) 文 [ESQL] [SP] 文, 695
WHERE 句
ALTER EVENT 文, 418
ALTER PUBLICATION 文, 435
CREATE EVENT 文, 542
CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 611
DELETE 文, 698
SELECT 文, 900
UPDATE 文, 979
UPDATE 文 [SQL Remote], 974
探索条件, 34
WHILE 句
LOOP 文, 832
WHILE 文
Transact-SQL 構文, 988

window-name
一般的な SQL 構文要素, 404
window-spec
Window 関数の構文, 989
Window 関数
AVG 関数, 156
COUNT_BIG 関数, 185
COUNT 関数, 183
COVAR_POP 関数, 187
CUME_DIST 関数, 191
DENSE_RANK 関数, 213
MAX 関数, 275
MEDIAN 関数, 276
MIN 関数, 278
PERCENT_RANK 関数, 300
RANK 関数, 311
REGR_AVGX 関数, 316
REGR_AVGY 関数, 317
REGR_COUNT 関数, 318
REGR_INTERCEPT 関数, 320
REGR_R2 関数, 321
REGR_SLOPE 関数, 322
REGR_SXX 関数, 324
REGR_SXY 関数, 326
ROW_NUMBER 関数, 336
STDDEV_POP 関数, 354
STDDEV_SAMP 関数, 356
STDDEV 関数, 354
SUM 関数, 363
VAR_POP 関数, 384
VAR_SAMP 関数, 386
WINDOW 句
SELECT 文, 900
構文, 989
WITH AUTO NAME 句
INSERT 文, 803
MERGE 文, 836
WITH BATCH 句
EXECUTE IMMEDIATE 文, 741
WITH CHECK OPTION 句
ALTER VIEW 文, 485
CREATE VIEW 文, 681
WITH CHECKPOINT LOG 句
BACKUP 文, 491
WITH CHECKPOINT 句
LOAD TABLE 文, 824
WITH COMMENT 句
BACKUP 文, 491
WITH CONTENT LOGGING 句
LOAD TABLE 文, 825
データベースミラーリングに必要, 826
WITH ESCAPES 句
EXECUTE IMMEDIATE 文, 740
WITH EXCLUSIVE MODE 句
REFRESH MATERIALIZED VIEW 文, 867
REFRESH TEXT INDEX 文, 871
WITH EXECUTE 句
PREPARE 文, 857
WITH EXPRESS CHECK 句
VALIDATE 文, 983
WITH FILE NAME LOGGING 句
LOAD TABLE 文, 824
WITH GRANT OPTION 句
構文, 785
WITH HOLD カーソル
DECLARE CURSOR 文 [ESQL] [SP], 686
OPEN 文, 845
ロック, 845
WITH HOLD 句
LOCK TABLE 文, 831
OPEN 文, 845
WITH ISOLATION LEVEL 句
REFRESH MATERIALIZED VIEW 文, 867
REFRESH TEXT INDEX 文, 870
WITH LOG 句
WRITETEXT 文 [T-SQL], 991
WITH MAX 句
ALLOCATE DESCRIPTOR 文 [ESQL], 407
WITH OPTION 句
SETUSER 文, 932
WITHOUT SAVE 句
DETACH TRACING 文, 707
WITH QUOTES 句
EXECUTE IMMEDIATE 文, 740
WITH RECOMPILE 句
CREATE PROCEDURE 文 [T-SQL], 599
WITH RECURSIVE 句
SELECT 文, 897
WITH RESULT SET 句
EXECUTE IMMEDIATE 文, 741
WITH ROW LOGGING 句
LOAD TABLE 文, 825
データベースミラーリングに必要, 826
WITH SAVE 句
DETACH TRACING 文, 707
WITH SCRIPTED UPLOAD 句

CREATE PUBLICATION 文 [Mobile Link] [SQL Remote], 610
WITH SERVER NAME 句
START DATABASE 文, 935
WITH SHARE MODE 句
REFRESH MATERIALIZED VIEW 文, 868
REFRESH TEXT INDEX 文, 871
WITH TEXTPTR 句
GET DATA 文, 771
WITH TRUNCATE AT CHECKPOINT 句
START DATABASE 文, 935
WITH VARIABLE RESULT 句
DESCRIBE 文 [ESQL], 702
PREPARE 文, 857
WITH 句
INTERSECT 文, 810
REFRESH MATERIALIZED VIEW 文, 867
REFRESH TEXT INDEX 文, 870
REVOKE REMOTE DBA 文 [SQL Remote], 893
SELECT 文, 897
WORK 句
COMMIT 文, 514
REVOKE REMOTE DBA 文 [SQL Remote], 890
WRITE_CLIENT_FILE 関数
構文, 390
WRITECLIENTFILE authority 句
GRANT 文, 784
WRITECLIENTFILE 権限
REVOKE 文, 888
WRITECLIENTFILE 権限
GRANT 文, 782
WriteNoPK ロック
sa_locks システムプロシージャー, 1112
WRITETEXT 文
Transact-SQL 構文, 991
WSDL
CREATE FUNCTION 文 [Web クライアント], 558
CREATE PROCEDURE 文 [Web クライアント], 598
CREATE SERVICE 文 [SOAP Web サービス], 447, 629
WWW-Authenticate
HTTP_RESPONSE_HEADER 関数, 251

X

XLOCK テーブルヒント
FROM 句, 764
XML
CREATE SERVICE 文 [HTTP Web サービス], 441, 623
openxml システムプロシージャー, 1035
XMLAGG 関数, 392
XMLCONCAT 関数, 393
XMLELEMENT 関数, 394
XMLFOREST 関数, 396
XMLGEN 関数, 398
XML データ型, 93
ロード、LOAD TABLE 文を使用, 821
XMLAGG 関数
構文, 392
XMLATTRIBUTES パラメーター
XMLELEMENT 関数, 394
XMLCONCAT 関数
構文, 393
XMLELEMENT 関数
構文, 394
XMLFOREST 関数
構文, 396
XMLGEN 関数
構文, 398
XML データ型
構文, 93
xp_cmdshell システムプロシージャー
構文, 1237
xp_msver システムプロシージャー
構文, 1238
xp_read_file システムプロシージャー
構文, 1239
xp_scanf システムプロシージャー
構文, 1240
xp_sendmail システムプロシージャー
構文, 1241
xp_sprintf システムプロシージャー
構文, 1245
xp_startmail システムプロシージャー
構文, 1246
xp_startsmtp システムプロシージャー
McAfee VirusScan での有効化, 1248
ウィルススキャナ設定と競合する可能性, 1248
構文, 1247
xp_stopmail システムプロシージャー
構文, 1249
xp_stopsmtp システムプロシージャー
構文, 1249

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

xp_write_file システムプロシージャー
構文, 1250

Y

YEARS 関数
構文, 399
YEAR 関数
構文, 399
YMD 関数
構文, 401

Z

zip ユーティリティ
COMPRESS 関数, 173

あ

アイドルサーバー
CREATE EVENT 文を使用してイベントを作成, 540
あいまい検索
CONTAINS 探索条件, 51
アサーション
正規表現, 28
正規表現の例, 28
アスタリスク
CONTAINS 句で使用できる構文, 54
サブクエリの使用, 682
全文クエリ文字列で使用できる構文, 54
値
プロシージャーから返す, 882
圧縮
ALTER TABLE 文を使用してテーブルを圧縮, 462
COMPRESS 関数, 173
統計, 1052
圧縮されたカラム
圧縮の統計情報の取得, 1048
圧縮済みカラム
ALTER TABLE 文, 462
アドレス
SQL Remote パブリッシャー, 436
アプリケーションのプロファイル
追跡レベルの設定, 1191
アポストロフィ
SQL 文字列内, 7
アルファベット
定義, 4
暗号化
CREATE DATABASE 文, 525
CREATE DECRYPTED DATABASE 文, 530
CREATE ENCRYPTED FILE 文, 537
SQL を使用してマテリアライズドビューを暗号化, 427
データベース、CREATE ENCRYPTED DATABASE 文, 535
暗号化アルゴリズム
CREATE DATABASE 文, 525
暗号化キー
暗号化されたデータベースのキーの変更, 540
アンロード
UNLOAD 文を使用した結果セット, 966
UNLOAD 文を使用したデータ, 966
コストモデル, 1206
アーカイブ
BACKUP 文を使用してデータベースバックアップを作成, 489
データベースのリストア, 880
アーカイブバックアップ
サポートされるオペレーティングシステム、BACKUP 文の使用, 489
アークコサイン関数
ACOS 関数, 152
アークサイン関数
ASIN 関数, 154
アークタンジェント関数
ATAN2 関数, 156
ATAN 関数, 155
アーティクル
SYSARTICLECOL システムビュー, 1255
SYSARTICLE システムビュー, 1254

い

依存性
sa_dependent_views システムプロシージャーを使用して決定, 1070
一意性
CREATE TABLE 文での制約, 658
位置付け DELETE 文
構文, 695
位置ロック
sa_locks システムプロシージャー, 1112
一致タイプ
参照整合性, 660
イベント

ALTER EVENT 文, 417
ALTER EVENT 文を使用したスケジューリング, 417
ALTER EVENT 文を使用した変更, 417
COMMENT 文を使用してコメントを追加, 512
CREATE EVENT 文, 540
CREATE EVENT 文を使用したスケジューリング, 540
DROP EVENT 文, 713
DROP EVENT 文を使用して削除, 713
EVENT_PARAMETER, 223
TRIGGER EVENT 文, 960
トリガー, 960
無効化, 417
イベント条件
リスト, 222
イベントハンドラー
ALTER EVENT 文を使用して隠す, 417
イメージ
データベースから読み込み, 866
イメージ、SQL 関数
説明, 151
イメージバックアップ
BACKUP 文を使用して作成, 489
インジケーター
コメント, 84
インストール
Java クラス, 809
インデックス
ALTER INDEX 文, 422
ALTER INDEX 文を使用したクラスタリング, 422
ALTER INDEX 文を使用した名前変更, 422
COMMENT 文を使用してコメントを追加, 512
CREATE INDEX 文, 567
DROP INDEX 文, 715
OLAP 負荷の最適化, 569
sa_index_density を使用してインデックス断片化を検出, 1101
sa_index_density を使用して無駄の多いインデックスを検出, 1101
SYSPHYSIDX システムビューに物理インデックスを記録, 1285
VALIDATE 文, 983
圧縮, 877
外部キー, 570
仮想, 568
関数, 569
組み込み関数, 567
システムビュー, 1273
自動作成, 570
所有者, 570
テーブルでの使用, 570
ビュー, 570
プライマリキー, 570
命名, 570
ユニーク, 568
レベル, 1103
インデックスの削除
DROP 文, 715
インデックスの作成
CREATE INDEX 文, 567
インデックスヒント
FROM 句, 766
引用符
ASE との互換性, 33
SQL 識別子, 4
一重と二重, 33
データベースオブジェクト, 4

う

ウィンドウ (OLAP)
WINDOW 句, 989
ウォッチリスト
sa_server_option を使用した設定, 1179
閏年
説明, 111

え

エイリアス
DELETE 文, 698
UPDATE 文, 979
カラム, 899
エクスポート
BLOB, 1250
UNLOAD 文を使用したデータのアンロード, 966
エスケープシーケンス
SQL 文字列内の 16 進値, 7
SQL 文字列内の Unicode 値, 7
SQL 文字列内の一重引用符, 7
SQL 文字列内の改行文字, 7
SQL 文字列内のバックスラッシュ, 7
Unicode, 380
エスケープ文字

INPUT 文, 794
OUTPUT 文, 848
説明, 7
バイナリリテラル, 6
エラー
CREATE EVENT 文を使用してイベントを作成, 540
Embedded SQL 内のエラーのトラップ, 987
RAISERROR 文, 862
通知, 934
ユーザー定義メッセージ, 1329
エラーメッセージ
ERRORMSG 関数, 218
円記号
SQL 識別子で使用できない, 4
SQL 文字列内, 7
エンコード
CREATE DATABASE 文, 524
LOAD TABLE 構文, 814
演算子
演算子の優先度, 12
算術演算子, 10
説明, 8
比較演算子, 8
ビット処理演算子, 11
文字列演算子, 11
論理演算子の説明, 9
演算子の優先度
構文, 12
全文検索, 53
演算の順序
SQL 演算子の優先度, 12
エンジン
データベースの起動, 938
データベースの停止, 949

お

大文字と小文字の区別
LIKE 探索条件, 45
REGEXP_SUBSTR 関数, 314
REGEXP 探索条件, 47
SIMILAR TO 探索条件, 49
比較演算子, 8
大文字の文字
UPPER 関数, 381
大文字の文字列
UCASE 関数, 378
UPPER 関数, 381
オブジェクトの置き換え
sa_make_object, 1115
オプション
Interactive SQL の設定, 916
quoted_identifier と T-SQL の互換性, 34
Remote オプションの設定, 919
SET OPTION 文, 913
sp_login_environment システムプロシージャーの初期設定, 1217
sp_tsql_environment システムプロシージャーの初期設定, 1227
sp_tsql_environment システムプロシージャーでの設定, 1227
SYSOPTIONS 統合ビュー, 1346
SYSUSEROPTIONS 統合ビュー, 1355
Transact-SQL の設定, 929
値を取得, 774
上書き, 1170
システムビュー, 1285
オプションウォッチリスト
sa_server_option を使用した設定, 1179
オプティマイザー
CREATE STATISTICS 文, 643
明示的な選択性推定, 60
オプティマイザー計画
テキスト仕様の取得, 747
オプティマイザーテーブル
説明, 1008
オプティマイザー統計情報
DROP STATISTICS 文を使用して削除, 727
オペレーティングシステム
コマンドの実行, 959
オーバーフローエラー
AVG 関数, 157
SUM 関数, 364

か

回帰関数
REGR_AVGX 関数, 316
REGR_AVGY 関数, 317
REGR_COUNT 関数, 318
REGR_INTERCEPT 関数, 320
REGR_R2 関数, 321
REGR_SLOPE 関数, 322
REGR_SXX 関数, 324
REGR_SXY 関数, 326

REGR_SYY 関数, 327
改行文字
SQL 文字列内, 7
開始
BEGIN TRANSACTION 文を使用してユーザー定義トランザクションを開始, 500
START EXTERNAL ENVIRONMENT 文を使用した外部環境の開始, 939
START JAVA 文を使用する Java VM, 940
サブスクリプション, 941
概数値データ型
説明, 93
外部オブジェクト
INSTALL EXTERNAL OBJECT 文を使用して作成, 807
REMOVE EXTERNAL OBJECT 文を使用した削除, 875
SYSEXTERNENVOBJECT システムビュー, 1266
外部環境
COMMENT 文を使用してコメントを追加, 512
START EXTERNAL ENVIRONMENT 文を使用した開始, 939
STOP EXTERNAL ENVIRONMENT 文を使用した停止, 947
外部関数
CREATE FUNCTION 文を使用したネイティブ関数へのインターフェイス, 550
CREATE PROCEDURE 文を使用したネイティブプロシージャーへのインターフェイス, 583
外部キー
ALTER INDEX 文, 422
ALTER INDEX 文を使用したクラスタリング, 422
ALTER INDEX 文を使用した名前変更, 422
COMMENT 文を使用してコメントを追加, 512
CREATE TABLE 文で名前なし, 659
CREATE TABLE 文での整合性の制約, 659
sp_remote_exported_keys システムプロシージャー, 1219
sp_remote_imported_keys システムプロシージャー, 1221
システムビュー, 1268
統合ビュー, 1344
外部参照
FROM 句, 762
ラテラル派生テーブル, 762
外部単語区切り
ALTER TEXT CONFIGURATION 文, 477
外部テーブル
システムビュー, 1268
外部プロシージャー
CREATE PROCEDURE 文を使用したネイティブプロシージャーへのインターフェイス, 583
外部ログイン
リモートサーバーへの外部ログインの削除, 713
リモートサーバーへの割り当て, 548
外部ログインの作成
CREATE EXTERNLOGIN 文, 548
解放
セーブポイント, 873
返されるロー数の制限
説明, 895
返す
プロシージャーから値, 882
書き込みチェックサム
検証, 983
角カッコ
SQL 識別子, 4
データベースオブジェクト, 4
拡張プロシージャー
説明, 1030
格納フォーマット
ALTER SPATIAL REFERENCE SYSTEM 文, 455
CREATE SPATIAL REFERENCE SYSTEM 文, 640
確立
セーブポイント, 895
型変換
説明, 127
傾き
回帰直線, 322
カタログ
システムテーブル, 993
システムテーブル内のデータの表示, 993
デフォルトのシステムビュー, 1253
カタログシステムプロシージャー
説明, 1029
カタログデータ
表示, 993
カタログプロシージャー
アルファベット順リスト, 1035
カタログプロシージャー (ASE)
sp_column_privileges, 1034

sp_pkeys, 1034
sp_special_columns, 1034
sp_sproc_columns, 1034
sp_statistics, 1034
sp_stored_procedures, 1034
sp_tables, 1034
Transact-SQL、リスト, 1034
カタログプロシージャー (ASE)
sp_columns, 1034
sp_fkeys, 1034
Transact-SQL リスト, 1033
カッコ
SQL 識別子, 4
データベースオブジェクト, 4
稼働サーバー
ALTER MIRROR SERVER 文, 430
CREATE MIRROR SERVER 文, 579
カラム
ALTER TABLE 文を使用した変更, 462
CREATE TABLE 文での制約, 658
SQL Remote 更新, 973
SYSTABCOL ビュー, 1315
エイリアス, 899
ドメイン, 126
ドメインの制約とデフォルト, 126
名前の変更, 472
バイナリデータの取得, 771
パーミッション, 1257
ユーザー定義データ型, 126
ロギングなしで更新, 991
カラム統計
LOAD TABLE を使用して部分的に更新, 827
SYSCOLSTATS 統合ビュー, 1343
SYSCOLSTAT システムビュー, 1258
選択性推定, 60
カラムの圧縮
ALTER TABLE 文, 462
CREATE TABLE 文, 651, 654
圧縮の統計情報の取得, 1048
カラムの制約
ALTER TABLE 文を使用して追加, 467
ALTER TABLE 文を使用して変更, 470
カラムの定義
CREATE TABLE 文, 654
カラムの統計情報
CREATE STATISTICS を使用した更新, 643
カラムパーミッションの更新
SYSCOLPERM システムビュー, 1257
カラム名
構文, 14
環境
ALTER EXTERNAL ENVIRONMENT 文を使用した変更, 420
関係
システムビュー, 1268
監査
sa_disable_auditing_type システムプロシージャーを使用した無効化, 1079
sa_enable_auditing_type システムプロシージャーを使用した有効化, 1082
コメントの追加, 1043
監視サーバー
ALTER MIRROR SERVER 文を使用した変更, 431
CREATE MIRROR SERVER 文を使用した定義, 580
関数
ALTER FUNCTION 文, 421
CREATE FUNCTION 文 [Web サービス] を使用して作成, 555
CREATE FUNCTION 文を使用したネイティブ関数へのインターフェイス, 550
CREATE PROCEDURE 文を使用したネイティブプロシージャーへのインターフェイス, 583
DROP FUNCTION 文を使用して削除, 714
HTTP, 147
Java, 144
SOAP, 147
SQL ストアド関数の作成, 563
イメージ、SQL, 151
インデックス, 569
外部関数インターフェイスの置き換え, 550
概要, 139
関数のタイプ, 139
システム, 150
集合, 139
数値, 146
その他, 146
テキスト、SQL, 151
テンポラリ, 563
データ型変換 SQL, 141
ネイティブ関数呼び出し, 551
日付と時刻, 142
ビット配列, 141
プロシージャーとの比較, 145
文字列, 148

ユーザー定義, 144
ユーザー定義関数から値を返す, 882
ユーザー定義関数から終了, 882
ランキング, 141
関数、HTTP
HTTP_HEADER, 249
HTTP_RESPONSE_HEADER, 251
HTTP_VARIABLE, 253
NEXT_HTTP_HEADER, 290
NEXT_HTTP_RESPONSE_HEADER, 291
NEXT_HTTP_VARIABLE, 292
関数、Java と SQL のユーザー定義
説明, 144
関数、SOAP
NEXT_SOAP_HEADER, 293
SOAP_HEADER, 346
関数、システム
DATALENGTH, 192
DB_EXTENDED_PROPERTY, 203
DB_ID, 207
DB_NAME, 208
DB_PROPERTY, 208
EVENT_CONDITION, 222
EVENT_CONDITION_NAME, 223
EVENT_PARAMETER, 223
ISENCRYPTED, 259
NEXT_CONNECTION, 288
NEXT_DATABASE, 289
PROPERTY, 304
PROPERTY_DESCRIPTION, 303
PROPERTY_NAME, 307
PROPERTY_NUMBER, 308
SUSER_ID, 365
SUSER_NAME, 365
TSEQUAL, 377
USER_ID, 381
USER_NAME, 382
関数、集合
AVG, 156
BIT_AND, 159
BIT_OR, 161
BIT_XOR, 163
COUNT, 183
COUNT_BIG, 185
FIRST_VALUE, 230
GROUPING, 238
LAST_VALUE, 262
LIST, 268
MAX, 275
MEDIAN, 276
MIN, 278
SET_BITS, 343
STDDEV, 354
STDDEV_POP, 354
STDDEV_SAMP, 356
SUM, 363
VAR_POP, 384
VAR_SAMP, 386
VARIANCE, 388
説明, 139
関数、数値
ABS, 151
ACOS, 152
ASIN, 154
ATAN, 155
ATAN2, 156
ATN2, 156
CEILING, 166
CONNECTION_EXTENDED_PROPERTY, 175
CONNECTION_PROPERTY, 178
COS, 182
COT, 183
DEGREES, 212
EXP, 226
FLOOR, 232
LOG, 272
LOG10, 273
MOD, 282
PI, 301
POWER, 303
RADIANS, 309
RAND, 310
REMAINDER, 328
ROUND, 335
SIGN, 344
SIN, 345
SQRT, 353
TAN, 367
TRUNCATE, 376
TRUNCNUM, 376
説明, 146
関数、その他
ARGN, 153
COALESCE, 170
CONFLICT, 173
ERRORMSG, 218

ESTIMATE, 219
ESTIMATE_SOURCE, 220
EXPERIENCE_ESTIMATE, 227
EXPLANATION, 228
GET_IDENTITY, 234
GRAPHICAL_PLAN, 236
GREATER, 238
IDENTITY, 254
IFNULL, 255
INDEX_ESTIMATE, 256
ISNUMERIC, 261
LESSER, 267
NEWID, 286
NULLIF, 295
NUMBER, 296
PLAN, 302
REWRITE, 332
SQLDIALECT, 352
TRACEBACK, 372
TRACED_PLAN, 373
TRANSACTSQL, 373
VAREXISTS, 387
WATCOMSQL, 388
説明, 146
関数、テキストとイメージ
TEXTPTR, 368
説明, 151
関数、データ型変換
CAST, 165
CONVERT, 179
HEXTOINT, 241
INTTOHEX, 257
ISDATE, 258
ISNULL, 260
説明, 141
関数、日付と時刻
DATE, 193
DATEADD, 194
DATEDIFF, 196
DATEFORMAT, 197
DATENAME, 198
DATEPART, 199
DATETIME, 200
DAY, 201
DAYNAME, 201
DAYS, 202
DOW, 215
GETDATE, 235
HOUR, 242
HOURS, 242
MINUTE, 280
MINUTES, 280
MONTH, 283
MONTHNAME, 283
MONTHS, 284
NOW, 294
QUARTER, 308
SECOND, 339
SECONDS, 340
SWITCHOFFSET, 366
SYSDATETIMEOFFSET, 367
TODATETIMEOFFSET, 371
TODAY, 372
WEEKS, 389
YEAR, 399
YEARS, 399
YMD, 401
説明, 142
関数、ビット
GET_BIT, 233
関数、ビット配列
BIT_LENGTH, 160
BIT_SUBSTR, 162
COUNT_SET_BITS, 186
SET_BIT, 341
アルファベット順リスト, 141
説明, 141
関数、文字列
ASCII, 153
BYTE_LENGTH, 164
BYTE_SUBSTR, 164
CHAR, 167
CHAR_LENGTH, 168
CHARINDEX, 169
COMPARE, 171
COMPRESS 関数, 172
CSCONVERT, 189
DECOMPRESS 関数, 209
DECRYPT 関数, 211
DIFFERENCE, 215
ENCRYPT 関数, 216
HASH 関数, 239
INSERTSTR, 256
LCASE, 264
LEFT, 265
LENGTH, 266

LOCATE, 271
LOWER, 273
LTRIM, 274
NCHAR, 286
PATINDEX, 297
READ_CLIENT_FILE, 313
REPEAT, 329
REPLACE, 330
REPLICATE, 331
REVERSE, 332
RIGHT, 334
RTRIM, 338
SIMILAR, 344
SORTKEY, 347
SOUNDEX, 350
SPACE, 351
STR, 358
STRING, 359
STRTOUUID, 359
STUFF, 361
SUBSTRING, 361
TO_CHAR, 369
TO_NCHAR, 370
TRIM, 375
UCASE, 378
UNICODE, 379
UNISTR, 380
UPPER, 381
UUIDTOSTR, 383
WRITE_CLIENT_FILE, 390
XMLAGG, 392
XMLCONCAT, 393
XMLELEMENT, 394
XMLFOREST, 396
XMLGEN, 398
説明, 148
関数、ランキング
説明, 141
カンマで区切ったリスト
LIST 関数、SQL 構文, 268
カーソル
CLOSE 文 [ESQL] [SP], 510
DESCRIBE 文 [ESQL], 700
EXPLAIN SQL 構文, 747
OPEN 文, 845
SELECT 文で設定される更新可能性, 900
Transact-SQL のカーソル、DECLARE CURSOR 文, 690
Transact-SQL の宣言、DECLARE CURSOR 文, 690
Transact-SQL の宣言、SELECT 文, 905
外部プロシージャーの再記述, 586
外部プロシージャーの動作の記述, 586
接続ごとのリスト, 1108
宣言, 686
データベースオプション設定の変更, 915
開く, 844
文の準備, 856
ユーザー定義プロシージャーでの再記述, 604
ユーザー定義プロシージャーの動作の記述, 604
ループ, 753
ローの削除, 695
ローの挿入, 860
ローのフェッチ, 749
カーソルの再記述
CREATE PROCEDURE 文, 604
CREATE PROCEDURE 文 [外部呼び出し], 586
カーソルを開く
OPEN 文, 844

き

記述
外部プロシージャーのカーソルの動作, 586
カーソル, 700
ユーザー定義プロシージャーのカーソルの動作, 604
記述子
DESCRIBE 文 [ESQL], 700
FETCH 文, 749
文の準備, 856
記述子領域
EXECUTE 文, 742
UPDATE (位置付け) 文, 971
情報の取得, 773
設定, 907
メモリの割り付け, 406
割り付けの解除, 684
規則
SQL 言語構文, 1
構文, 404
起動
CALL 文を使用してプロシージャーを起動, 503

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CREATE EVENT 文を使用してイベントを作成, 540
Interactive SQL のロギング, 940
データベース, 935
データベースサーバー, 938
データベース抽出中の SQL Remote サブスクリプション, 874
パススルーモード, 855
機能
SYSCAPABILITY システムビュー, 1255
リモートサーバー, 1256
ロック, 830
逆引用符
SQL 識別子, 4
データベースオブジェクト, 4
逆扁平率
定義済み, 1302
キャッシュ
フラッシュ, 1085
競合
SQL Remote の CONFLICT 関数, 173
競合の解決
SQL Remote の CONFLICT 関数, 173
協定世界時
CURRENT UTC TIMESTAMP, 66
UTC TIMESTAMP, 75
共用体
複数の SELECT 文, 964
強力な暗号化
CREATE DATABASE 文, 525
曲面モデル
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
拒否
パーミッションの付与, 887
許容値
空間計算の許容値の設定, 639
空間計算の許容値の変更, 454
近接検索
CONTAINS 探索条件, 51
キー
CREATE SEQUENCE 文, 616
キーワード
sa_reserved_words システムプロシージャー, 1161
SQL 予約語のリスト, 1
構文での SQL 予約語の使用方法, 1

く

句
順序、SQL の表記規則, 404
空間参照系
COMMENT 文を使用してコメントを追加, 512
事前に定義された機能のインストール, 1106
空間データ
ALTER SPATIAL REFERENCE SYSTEM 文, 453
CREATE SPATIAL REFERENCE SYSTEM 文, 635
CREATE SPATIAL UNIT OF MEASURE 文, 642
DROP SPATIAL REFERENCE SYSTEM 文, 724
DROP SPATIAL UNIT OF MEASURE 文, 725
ESRI シェイプファイルのインポート, 821
ISYSSPATIALREFERENCESYSTEM システムテーブル, 1004
ISYSUNITOFMEASURE システムテーブル, 1007
st_geometry_load_shapefile システムプロシージャー, 1235
TREAT 関数, 374
ジオメトリの分解, 1228
無効なジオメトリのトラブルシューティング, 1228
区切られた文字列
ASE との互換性, 33
クライアントでの文のキャッシュ
ReqeustLogFile プロパティ, 1181
クライアントファイル
READ_CLIENT_FILE 関数, 313
WRITE_CLIENT_FILE 関数, 390
クラス
Java クラスの削除, 876
Java メソッド, 144
クラスタードインデックス
ALTER INDEX 文を使用して作成, 422
クリア
Interactive SQL ウィンドウ枠, 510
繰り返し可能読み出し
FROM 句, 764
グリッドサイズ
ALTER SPATIAL REFERENCE SYSTEM 文, 454

CREATE SPATIAL REFERENCE SYSTEM 文, 639
グリッドにスナップ
SNAP TO GRID 句、ALTER SPATIAL REFERENCE SYSTEM 文, 454
SNAP TO GRID 句、CREATE SPATIAL REFERENCE SYSTEM 文, 639
グループ
GRANT 文, 782
グループ化
BEGIN 文で文をグループ化, 497
BEGIN 文で文をグループ化 [TSQL], 499
GROUP BY 句, 788
正規表現, 18
グローバルチェックサム
検証, 983
データベース設定の変更, 410
データベースの作成, 523
グローバルテンポラリテーブル
CREATE TABLE 文, 651
グローバル変数
@@identity, 83
アルファベット順リスト, 79
定義, 76
トリガーと @@identity, 83
グローバルユニーク識別子
NEWID 関数の SQL 構文, 286

け

計画
テキスト仕様の取得, 747
結果セット
UNLOAD 文を使用したデータのアンロード, 966
ストアドプロシージャーからの選択, 761
複数の結果セットの取得, 881
プロシージャーの実行の再開, 881
変数、CREATE PROCEDURE 文 [外部呼び出し], 585
変数、CREATE PROCEDURE 文, 604
変数、DESCRIBE 文 [ESQL], 702
変数、PREPARE 文, 857
結合
複数の SELECT 文の結果, 964
決定係数
説明, 321
決定的な動作
外部関数, 551
ユーザー定義関数, 564
権限
GRANT 文, 782
REMOTE DBA の付与, 778
言語の要素
構文, 1
検証
sa_validate システムプロシージャー, 1210
VALIDATE TABLE 文を使用するテーブル, 983
VALIDATE 権限, 784
VALIDATE 文, 983
VALIDATE 文を使用してインデックスを検証, 983
チェックサム, 983
パスワード, 1212

こ

高可用性
ALTER MIRROR SERVER 文, 430
CREATE MIRROR SERVER 文, 579
交差
複数の SELECT 文の結果, 810
更新
SQL Remote ジョイン, 974
SQL Remote テーブルとカラム, 973
ジョインに基づく, 979
パブリケーションとサブスクリプション, 980
ロギングなしでカラムを更新, 991
ロー, 976
構文
3 値的論理, 59
CASE 式, 16
CURRENT_TIMESTAMP 特別値, 64
CURRENT_USER 特別値, 65
CURRENT DATABASE 特別値, 62
CURRENT DATE 特別値, 62
CURRENT PUBLISHER 特別値, 62
CURRENT TIMESTAMP 特別値, 64
CURRENT USER 特別値, 65
CURRENT UTC TIMESTAMP 特別値, 66
IF 式, 15
IS NULL 探索条件, 57
IS TRUE または FALSE 探索条件, 58
LAST USER 特別値, 67
LEN 関数, 266

NULL 値, 67
SQLCODE 特別値, 70
SQL CURRENT REMOTE USER 特別値, 63
SQL CURRENT TIME 特別値, 63
SQLSTATE 特別値, 71
SQL 演算子, 8
SQL 演算子の優先度, 12
SQL 関数, 139
SQL キーワード, 1
SQL サブクエリ, 15
SQL 識別子, 4
SQL の式, 12
SQL 文, 406
SQL 変数, 76
SQL 予約語のリスト, 1
TIMESTAMP 特別値, 73
Transact-SQL 式の互換性, 33
USER 特別値, 74
UTC TIMESTAMP 特別値, 75
カラム名, 14
規格の準拠のテスト, 352
コメント, 84
算術演算子, 10
式内の定数, 14
述部, 34
接続レベル変数, 78
探索条件, 34
探索条件内の SQL サブクエリ, 36
定数, 6
特別値, 61
比較演算子, 8
ビット処理演算子, 11
表記規則, 404
マニュアルの表記規則, 403
文字列, 5
文字列演算子, 11
論理演算子, 9
ローカル変数, 77
構文の表記規則
SQL 文, 404
互換
ビュー, 1356
互換性
NULL, 69
Transact-SQL グローバル変数, 79
Transact-SQL 式, 33
Transact-SQL 比較演算子, 8
Transact-SQL ビュー, 1365
Transact-SQL ローカル変数, 77
T-SQL 式と QUOTED IDENTIFIER オプション, 34
日付と時刻, 132
互換ビュー
SYSCOLLATION, 1356
SYSCOLLATIONMAPPINGS, 1357
SYSCOLUMN, 1357
SYSFKCOL, 1358
SYSFOREIGNKEY, 1359
SYSINDEX, 1359
SYSINFO, 1360
SYSIXCOL, 1361
SYSTABLE, 1361
SYSUSERLIST, 1363
SYSUSERPERM, 1363
SYSUSERPERMS, 1364
説明, 1356
国際言語と文字セット
置換文字, 128
コサイン関数
COS 関数, 182
コストベースの最適化
FORCE NO OPTIMIZATION 句を使用して回避, 902
FORCE OPTIMIZATION オプションを使用して強制実行, 902
プロシージャーに対する強制実行, 902
コストモデル
ALTER DATABASE 文を使用した再調整, 408
アンロード, 1206
データベースサーバーの調整, 408
ローディング, 1109
コタンジェント関数
COT 関数, 183
コピーサーバー
ALTER MIRROR SERVER 文を使用した変更, 431
CREATE MIRROR SERVER 文を使用した定義, 581
コピーノード
サーバーの削除, 719
コマンド
オペレーティングシステムコマンドの実行, 959
コミット
2 フェーズの準備, 858

COMMIT 文を使用してトランザクションをコミット, 514
コミットされた読み出し
FROM 句, 764
コメント
COMMENT 文を使用してデータベースオブジェクトに追加, 512
構文, 84
小文字の文字列
LCASE 関数, 264
LOWER 関数, 273
混合格納フォーマット
ALTER SPATIAL REFERENCE SYSTEM 文, 455
CREATE SPATIAL REFERENCE SYSTEM 文, 640
コード化
UNLOAD 文, 966
コードページ
INPUT 文, 796
OUTPUT 文, 849

さ

再開
プロシージャーの実行, 881
最小
データの範囲, 112
最大
データの範囲, 112
最適化
FORCE NO OPTIMIZATION 句を使用して回避, 902
FORCE OPTIMIZATION オプションを使用して強制実行, 902
既存のテーブルの定義, 547
最適化のバイパス
FORCE NO OPTIMIZATION 句を使用して回避, 902
FORCE OPTIMIZATION オプションを使用して回避, 902
削除
ALTER TABLE 文を使用してカラムを削除, 462
DROP CONNECTION 文を使用して接続を削除, 709
DROP DATABASE 文を使用してデータベースファイルを削除, 709
DROP DBSPACE 文を使用して DB 領域を削除, 711
DROP DOMAIN 文を使用してドメインを削除, 712
DROP EVENT 文を使用してイベントを削除, 713
DROP FUNCTION 文を使用して関数を削除, 714
DROP FUNCTION 文を使用してトリガーを削除, 714
DROP INDEX 文を使用してインデックスを削除, 715
DROP LOGIN POLICY 文を使用してログインポリシーを削除, 716
DROP MATERIALIZED VIEW 文を使用してマテリアライズドビューを削除, 717
DROP MESSAGE 文を使用してメッセージを削除, 718
DROP PROCEDURE 文を使用してプロシージャーを削除, 720
DROP PUBLICATION 文, 721
DROP SERVER 文を使用してリモートサーバーを削除, 723
DROP SERVICE 文を使用して Web サービスを削除, 724
DROP STATEMENT 文を使用して準備文を削除, 726
DROP STATISTICS 文を使用してオプティマイザー統計情報を削除, 727
DROP SUBSCRIPTION 文, 728
DROP SYNCHRONIZATION SUBSCRIPTION 文, 729
DROP SYNCHRONIZATION USER 文, 731
DROP TABLE 文を使用してテーブルを削除, 731
DROP USER 文を使用してログインポリシーを削除, 735
DROP VARIABLE 文を使用して SQL 変数を削除, 736
DROP VIEW 文を使用してビューを削除, 737
DROP 文を使用してトリガーを削除, 734
Java クラス, 876
REVOKE 文を使用してユーザーを削除, 887
START SYNCHRONIZATION DELETE 文, 943
STOP SYNCHRONIZATION DELETE 文, 951
カーソルからローを削除, 695
空間参照系、DROP SPATIAL REFERENCE SYSTEM 文の使用, 724

全文検索用のテキストインデックス, 733
テキスト設定オブジェクト, 733
データベースのロー, 697
テーブルからすべてのロー, 961
ドメイン, 712
パーミッション, 887
パーミッションの付与, 887
リモートサーバーへのログイン, 713
リモートメッセージタイプ, 722
作成
BACKUP 文を使用してデータベースバックアップを作成, 489
CREATE DATABASE 文を使用してデータベースを作成, 521
CREATE DBSPACE 文を使用してデータベースファイルを作成, 528
CREATE EXISTING TABLE 文を使用してプロキシテーブルを作成, 546
CREATE INDEX 文, 567
CREATE LOCAL TEMPORARY TABLE 文を使用してローカルテンポラリファイルを作成, 571
CREATE MATERIALIZED VIEW 文, 576
CREATE PUBLICATION 文, 610
CREATE SYNCHRONIZATION PROFILE 文, 646
CREATE SYNCHRONIZATION SUBSCRIPTION 文, 647
CREATE TABLE 文, 651
CREATE TRIGGER 文, 670
CREATE USER 文を使用してユーザーを作成, 677
CREATE VARIABLE 文を使用して SQL 変数を作成, 679
CREATE VIEW 文, 681
DECLARE 文を使用して SQL 変数を作成, 693
sp_remote_tables システムプロシージャーを使用してプロキシテーブルを作成, 1226
SQL Remote リモートメッセージタイプ, 613
Transact-SQL のトリガー, 676
カーソル, 686
サブスクリプション, 645
サーバー, 618
スキーマ, 615
ストアドプロシージャー, 602
全文検索用のテキストインデックス, 667
全文検索用のテキスト設定オブジェクト, 666
セーブポイント, 895
プロキシテーブル, 653
メッセージ, 578
ローカルテンポラリテーブル, 691
サブクエリ
SQL 探索条件内, 36
一致するローがない場合は NULL と評価, 15
構文, 15
サブスクリプション
ALTER SYNCHRONIZATION SUBSCRIPTION 文, 458
CREATE SUBSCRIPTION 文 [SQL Remote], 645
CREATE SYNCHRONIZATION SUBSCRIPTION 文, 647
DROP SUBSCRIPTION 文, 728
DROP SYNCHRONIZATION SUBSCRIPTION 文, 729
SQL Remote REMOTE RESET 文, 874
SQL Remote UPDATE 文, 975
START SUBSCRIPTION 文 [SQL Remote], 941
STOP SUBSCRIPTION 文 [SQL Remote], 950
SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote], 958
UPDATE 文, 980
サブスクリプションの停止
STOP SUBSCRIPTION 文, 950
サブスクリプションの同期
SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote], 958
算術演算子
SQL 構文, 10
モジュロ, 10
参照整合性
CREATE TABLE 文の Match 句, 660
FROM 句, 761
サンプル変数
説明, 386
サーバー
ALTER SERVER 文を使用してリモート属性を変更, 439
CREATE EVENT 文を使用してイベントを作成, 540
作成, 618
データベースの起動, 938
データベースの停止, 949
リモートサーバーの削除, 723
サーバーオプション
sa_server_option システムプロシージャーでの設定, 1170

サービス
ALTER SERVICE 文 [HTTP Web サービス] を使用して HTTP サービスを変更, 441
ALTER SERVICE 文 [SOAP Web サービス] を使用して SOAP サービスを変更, 446
COMMENT 文を使用してコメントを追加, 512
DROP SERVICE 文を使用して Web サービスを削除, 724

し

シェイプファイル
INPUT 文を使用したロード, 795
LOAD TABLE 文を使用したロード, 821
st_geometry_load_shapefile システムプロシージャーを使用するロード, 1235
ジオメトリ
st_geometry_dump でのトラブルシューティング, 1228
式
CASE 式, 16
IF 式, 15
SQL 演算子の優先度, 12
Transact-SQL の互換性, 33
カラム名, 14
構文, 12
サブクエリ, 15
算術演算子, 10
説明, 12
定数, 14
データ型, 229
文字列演算子, 11
式の互換性
説明, 33
識別子
SQL Anywhere での最大長, 4
構文, 4
説明, 4
時刻
格納, 106
クエリ, 111
データベースへの送信, 106
比較, 130
変換関数, 142
時刻関数
アルファベット順リスト, 142
時刻データ型
DATETIME, 113
DATETIMEOFFSET, 114
SMALLDATETIME, 116
TIMESTAMP, 118
TIMESTAMP WITH TIME ZONE, 120
概要, 105
指数関数
EXP 関数, 226
システム拡張プロシージャー
説明, 1030
システムカタログ
テーブル, 993
ビュー, 1253
システム関数
アルファベット順リスト, 150
互換性, 150
システムテーブル
DUMMY, 993
Java, 1027
RowGenerator, 1027
システムテーブルデータの表示, 993
説明, 993
ビュー, 993
システムビュー
SYSARTICLE, 1254
SYSARTICLECOL, 1255
SYSCAPABILITY, 1255
SYSCAPABILITYNAME, 1256
SYSCHECK, 1256
SYSCOLPERM, 1257
SYSCOLSTAT, 1258
SYSCONSTRAINT, 1259
SYSDBFILE, 1259
SYSDBSPACE, 1260
SYSDBSPACEPERM, 1261
SYSDEPENDENCY, 1262
SYSDOMAIN, 1262
SYSEVENT, 1263
SYSEVENTTYPE, 1264
SYSEXTERNENV, 1265
SYSEXTERNENVOBJECT, 1266
SYSEXTERNLOGIN, 1267
SYSFILE, 1358
SYSFKEY, 1268
SYSFOREIGNKEYS, 1344
SYSGROUP, 1269
SYSHISTORY, 1269
SYSIDX, 1273
SYSIDXCOL, 1275

SYSJAR, 1276
SYSJARCOMPONENT, 1277
SYSJAVACLASS, 1277
SYSLOGINMAP, 1278
SYSLOGINPOLICY, 1279
SYSLOGINPOLICYOPTION, 1280
SYSMIRROROPTION, 1280
SYSMIRRORSERVER, 1280
SYSMIRRORSERVEROPTION, 1281
SYSMVOPTION, 1282
SYSMVOPTIONNAME, 1283
SYSOBJECT, 1283
SYSOPTION, 1285
SYSOPTSTAT, 1285
SYSPHYSIDX, 1285
SYSPROCEDURE, 1286
SYSPROCPARM, 1288
SYSPROCPERM, 1290
SYSPROXYTAB, 1291
SYSPUBLICATION, 1291
SYSREMARK, 1293
SYSREMOTEOPTION, 1293
SYSREMOTEOPTIONTYPE, 1294
SYSREMOTETYPE, 1294
SYSREMOTEUSER, 1295
SYSSCHEDULE, 1297
SYSSEQUENCE, 1299
SYSSEQUENCEPERM, 1300
SYSSERVER, 1301
SYSSOURCE, 1301
SYSSPATIALREFERENCESYSTEM, 1302
SYSSQLSERVERTYPE, 1306
SYSSUBSCRIPTION, 1307
SYSSYNC, 1308
SYSSYNCPROFILE, 1309
SYSSYNCSCRIPT, 1310
SYSTAB, 1311
SYSTABCOL, 1315
SYSTABLEPERM, 1318
SYSTEXTCONFIG, 1320
SYSTEXTIDX, 1322
SYSTEXTIDXTAB, 1324
SYSTRIGGER, 1324
SYSTYPEMAP, 1327
SYSUNITOFMEASURE, 1327
SYSUSER, 1328
SYSUSERAUTHORITY, 1329
SYSUSERMESSAGE, 1329
SYSUSERTYPE, 1330
SYSVIEW, 1331
SYSWEBSERVICE, 1333
システムビューのアルファベット順リスト, 1253
説明, 1253
表示, 1253
システムプロシージャー
Adaptive Server Enterprise システムプロシージャー, 1033
HTTP, 1030
SOAP, 1030
Sybase Central, 1029
Transact-SQL, 1033
Transact-SQL リスト, 1033
アルファベット順リスト, 1035
拡張リスト, 1030
説明, 1029
定義の表示, 1029
メッセージの作成, 578
呼び出すときに許可される名前付き, 503
システムプロシージャーとカタログストアドプロシージャー
説明, 1035
システム呼び出し
xp_cmdshell システムプロシージャー, 1237
ストアドプロシージャー, 1237
事前フィルター
ALTER TEXT CONFIGURATION 文, 478
ALTER TEXT CONFIGURATION 文を使用した削除, 478
実行
Transact-SQL でのストアドプロシージャー, 744
オペレーティングシステムコマンド, 959
準備文, 742
ファイルから SQL 文, 863
プロシージャーの実行の再開, 881
実行時間
START LOGGING 文, 940
実行プラン
プランをファイルに保存する例, 237
集合演算子
EXCEPT 文, 738
INTERSECT 文, 810
UNION 文, 964
集合共用体, 964
集合差, 738

集合積, 810
集合関数
アルファベット順リスト, 139
従属変数
回帰直線, 317
週日
ISO 8601, 107
終了
Interactive SQL, 746
トランザクションのロールバック, 890
プロシージャー, 882
終了コード
EXIT 文 [Interactive SQL], 746
述部
3 値的論理, 59
ALL 探索条件, 37
ANY 探索条件, 38
BETWEEN 探索条件, 39
CONTAINS 探索条件, 51
EXISTS 探索条件, 57
IN 探索条件, 50
IS DISTINCT FROM, 39
IS NOT DISTINCT FROM, 39
IS NOT NULL 探索条件, 57
IS NULL 探索条件, 57
IS TRUE または FALSE 探索条件, 58
IS UNKNOWN 探索条件, 58
LIKE 探索条件, 42
REGEXP 探索条件, 47
SIMILAR TO 探索条件, 48
SOME 探索条件, 38
構文, 34
述部内の SQL サブクエリ, 36
説明, 34
比較演算子, 8
明示的な選択性推定, 60
出力
メッセージウィンドウにメッセージを出力, 859
手動ビュー
ALTER MATERIALIZED VIEW 文, 427
取得
オプション値, 774
カラムからバイナリデータを取得, 771
記述子領域から情報を取得, 773
長いカラム名, 702
複数の結果セット, 881
準備
2 フェーズコミット, 858
準備文
DROP STATEMENT 文を使用して削除, 726
実行, 742
ジョイン
ANSI への書き換え, 332
FROM 句 SQL 構文, 758
ジョインに基づく SQL Remote 更新, 974
ジョインに基づく更新, 979
ジョインに基づくローの削除, 697
ジョイン演算子
ASE との互換性, 12
条件
3 値的論理, 59
CONTAINS, 51
EXISTS, 57
SQL 探索条件, 34
検索, 34
サブクエリ内, 36
照合
REGEXP_SUBSTR 関数, 314
REGEXP 探索条件, 47
SIMILAR TO 探索条件, 49
SORTKEY 関数, 347
データベースの作成時に適合化, 524
照合順
CREATE DATABASE 文, 524
LIKE 探索条件, 43
照合の適合化
COLLATION 句、CREATE DATABASE 文, 524
COMPARE 関数, 171
NCHAR COLLATION 句、CREATE DATABASE 文, 526
SORTKEY 関数, 347
商標情報
取り出し, 1238
初期化
CREATE DATABASE 文を使用してデータベースを初期化, 521
処理
Embedded SQL 内のエラー, 987
RAISERROR 文, 862
診断
sa_performance_statistics システムプロシージャー, 1149
診断トレーシング
ATTACH TRACING 文, 487
DETACH TRACING 文, 706

REFRESH TRACING 文, 872
sa_diagnostic_auxiliary_catalog テーブル, 1009
sa_diagnostic_blocking テーブル, 1009
sa_diagnostic_cachecontents テーブル, 1011
sa_diagnostic_connection テーブル, 1012
sa_diagnostic_cursor テーブル, 1013
sa_diagnostic_deadlock テーブル, 1014
sa_diagnostic_hostvariable テーブル, 1015
sa_diagnostic_internalvariable テーブル, 1016
sa_diagnostic_query テーブル, 1017
sa_diagnostic_request テーブル, 1019
sa_diagnostic_statement テーブル, 1021
sa_diagnostic_statistics テーブル, 1021
sa_diagnostic_tracing_level テーブル, 1022
sa_save_trace_data システムプロシージャー, 1165
sa_set_tracing_level システムプロシージャー, 1191
テーブル、説明, 1008
レコードの削除, 1159
診断トレーシングレベル
コマンドラインでの設定, 1191
シーケンス
ALTER SEQUENCE 文, 437
COMMENT 文を使用してコメントを追加, 512
CREATE SEQUENCE 文, 616
CREATE TABLE 文, 465, 656
DROP SEQUENCE 文, 722
SELECT 文, 903
使用パーミッションの取り消し, 888
使用パーミッションの付与, 783
シーケンスジェネレーター
ALTER SEQUENCE 文, 437
COMMENT 文を使用してコメントを追加, 512
CREATE SEQUENCE 文, 616
DROP SEQUENCE 文, 722
使用パーミッションの取り消し, 888

す

推定
明示的な選択性, 60
数値関数
アルファベット順リスト, 146
数値式
算術演算子, 10
数値データ型
BIGINT, 94
BIT, 95
DECIMAL, 96
DOUBLE, 97
DOUBLE を NUMERIC に変換する, 135
FLOAT, 98
INTEGER, 99
NUMERIC, 99
REAL, 101
SMALLINT, 101
TINYINT, 102
説明, 93
数値データ型のアルファベット順リスト
説明, 93
スキュー
sa_index_density を使用してインデックス内のスキューを検出, 1101
スキーマ
作成, 615
システムテーブル, 993
デフォルトのシステムビュー, 1253
スキーマロック
sa_locks システムプロシージャー, 1112
スクリプト化されたアップロード
CREATE PUBLICATION 構文, 610
進行状況値を TIMESTAMP に変換, 1063
進行状況値を UNSIGNED BIGINT に変換, 1063
スケジューリング
ALTER EVENT 文を使用してイベントをスケジューリング, 417
CREATE EVENT 文を使用してイベントを作成, 540
CREATE EVENT 文を使用してイベントをスケジューリング, 540
スケジュール
WAITFOR, 985
スケジュールされたイベント
WAITFOR 文, 985
トリガー, 960
スケールアウト
SET MIRROR OPTION 文, 908
サーバーの削除, 719
ステータス情報ファイル
ALTER MIRROR SERVER 文, 432
CREATE MIRROR SERVER 文, 582
ストアド関数
ネイティブ関数呼び出し, 551
ストアドプロシージャー

INPUT 文を使用できない, 800
Transact-SQL の作成, 599
Transact-SQL の実行, 744
T-SQL の変換, 388
Web サービスの作成, 591
作成, 602
システムプロシージャー, 1029
選択, 761
動的 SQL の実行, 740
ネイティブ関数呼び出し, 587
ストップリスト
CREATE STOPLIST 句, 477
STOPLIST 句, 477
スナップショット
BEGIN SNAPSHOT 文, 496
スナップショットアイソレーション
BEGIN SNAPSHOT 文, 496
sa_snapshots システムプロシージャー, 1193
sa_transactions システムプロシージャー, 1205
全文検索での使用, 668
スラッシュ - アスタリスク
コメントインジケーター, 84
スーパータイプ
説明, 127

せ

正規表現
REGEXP_SUBSTR 関数, 314
REGEXP 構文, 47
REGEXP 探索条件, 18
SIMILAR TO 探索条件, 48
アサーションのリスト, 28
アサーションの例, 28
構文, 18
サポートされている量指定子, 18
説明, 17
データベースの照合と一致, 40
特殊文字クラス, 22
メタ文字, 18
例, 31
ワイルドカード、グループ化、セット, 18
制御文
BREAK 構文, 502
CALL 文, 503
CASE 文, 505
CASE 文 [T-SQL], 507
CONTINUE 文の構文, 520
GOTO Transact-SQL 文, 775
IF 文, 791
LEAVE 文, 812
LOOP 文, 832
Transact-SQL CONTINUE 文, 988
Transact-SQL IF 文, 793
Transact-SQL WHILE 文, 988
WHILE 文, 832
整合性
CREATE TABLE 文での制約, 658
整数
テーブルの生成, 1165
静的カーソル
宣言, 686
製品名
取り出し, 1238
制約
カラム、CREATE TABLE 文, 658
名前の変更, 472
セキュリティ
SQL Remote レプリケーション, 886
レプリケーション, 778
接続
CONNECT 文を使用してデータベースに接続, 516
CREATE EVENT 文を使用してイベントを作成, 540
DROP CONNECTION 文, 709
Interactive SQL の切断, 707
RAISERROR による不許可, 863
カーソルのリスト, 1108
最大数の設定, 863
サーバーへの接続の無効化, 1170
失敗した接続のイベントの作成, 540
接続 ID リストの生成, 1059
設定, 906
プールの有効化, 932
接続の削除
DROP CONNECTION 文, 709
接続の無効化
個々のデータベース, 1178
データベース上の全データベース, 1178
接続レベル変数
構文, 78
定義, 76
切断
CREATE EVENT 文を使用してイベントを作成, 540

DROP CONNECTION 文, 709
Interactive SQL 接続, 707
設定
Interactive SQL のオプション, 916
Remote オプション, 919
SQLCA, 926
SQL 変数の値, 927
Transact-SQL のオプション, 929
オプション, 913
記述子領域, 907
接続, 906
ユーザー, 932
セット
正規表現, 18
セット演算
NULL, 69
宣言
DECLARE 文を使用した変数, 693
Embedded SQL ホスト変数, 685
Transact-SQL カーソル、SELECT 文, 905
カーソル, 686
選択
UNLOAD 文を使用したアンロードのための, 966
共通部分の形成, 810
共用体の形成, 964
セットの差の形成, 738
ロー, 895
選択性推定
推定ソース, 220
ユーザー定義, 60
線の解釈、空間データ
曲面タイプ, 637
平面タイプ, 637
全文検索
スナップショットアイソレーションの互換性, 668
全文検索
ALTER TEXT CONFIGURATION 文, 476
ALTER TEXT INDEX 文, 480
CONTAINS 探索条件, 51
CREATE TEXT CONFIGURATION 文, 666
CREATE TEXT INDEX 文, 667
DROP TEXT CONFIGURATION 文, 733
DROP TEXT INDEX 文, 733
FROM 句の中の CONTAINS 句, 758
REFRESH TEXT INDEX 文, 870
sa_char_terms システムプロシージャー, 1044
sa_nchar_terms システムプロシージャー, 1139
sa_refresh_text_indexes システムプロシージャー, 1158
sa_text_index_stats システムプロシージャー, 1201
sa_text_index_vocab_nchar システムプロシージャー, 1204
sa_text_index_vocab システムプロシージャー, 1202
TRUNCATE TEXT INDEX 文, 963
アスタリスク、使用できる構文, 54
演算子の優先度, 53
クエリ文字列で英数字以外の文字を使用する場合の注意, 53
特殊文字を使用できる構文, 55
ハイフン、使用できる構文, 55
セーブポイント
解放, 873
作成, 895
セーブポイントへロールバック, 891

そ

相関関数
CORR 関数, 181
相関名
DELETE 文, 698
UPDATE 文, 979
送信
リモートサーバーへ SQL 文を送信, 757
相対パス
INPUT 文, 794
READ 文, 863
挿入
LOAD TABLE 文を使用したデータの挿入, 814
SET 文を使用して BLOB を挿入, 927
xp_read_file システムプロシージャーを使用して BLOB を挿入, 1239
カーソルを使用したローの挿入, 860
テーブルへローを挿入, 802
マルチロー, 743
ローのバルク挿入, 814
ワイド挿入, 743
即時ビュー
ALTER MATERIALIZED VIEW 文, 427
基本となるテーブルの更新に必要のないパーミッション, 428
属性

ALTER SERVER 文を使用してリモートサーバー属性を変更, 439
測定単位
ALTER SPATIAL REFERENCE SYSTEM 文, 453
COMMENT 文を使用してコメントを追加, 512
CREATE SPATIAL REFERENCE SYSTEM 文, 635
CREATE SPATIAL UNIT OF MEASURE 文, 642
ST_UNITS_OF_MEASURE システムビュー, 1340
事前に定義された機能のインストール, 1106
速度プロパティ
値の設定, 1209
値の増分, 1208
損失を伴う変換
説明, 128
ソート
SORTKEY 関数, 347
ソートキー
SORTKEY 関数を使用した生成, 347

た

代替サーバー名
CREATE MIRROR SERVER 文を使用した定義, 580
代入
SQL 変数への値の代入, 927
タイプ式
探索条件, 34
タイムスタンプ
TIMESTAMP WITH TIMEZONE を使用する比較, 121
TIMESTAMP WITH TIMEZONE を使用する変換, 121
比較, 130
タイムゾーン
CURRENT UTC TIMESTAMP, 66
UTC TIMESTAMP, 75
ダウンロードのみ
CREATE PUBLICATION 構文, 610
多角形のフォーマット
Clockwise, 640
CounterClockwise, 640
EvenOdd, 640
多角形の方向
Clockwise, 640
CounterClockwise, 640
EvenOdd, 640
単語
MAXIMUM TERM LENGTH 句, 477
MINIMUM TERM LENGTH 句, 477
TERM BREAKER 句, 477
単語区切り
sa_char_terms での文字列を単語に分割する方法のテスト, 1044
sa_nchar_terms での文字列を単語に分割する方法のテスト, 1139
クエリ文字列で英数字以外の文字を使用する場合の注意, 53
単語区切りの外部ライブラリの指定, 477
探索条件
3 値的論理, 59
ALL, 37
ANY, 38
BETWEEN, 39
CONTAINS, 51
EXISTS, 57
IN, 50
IS DISTINCT FROM, 39
IS NOT DISTINCT FROM, 39
IS NOT NULL, 57
IS NULL, 57
IS TRUE または FALSE 探索条件, 58
IS UNKNOWN 探索条件, 58
LIKE, 42
REGEXP, 47
SIMILAR TO, 48
SOME, 38
構文, 34
サブクエリ内, 36
真理値, 58
説明, 34
明示的な選択性推定, 60
断片化
REORGANIZE TABLE での断片化解除, 877
sa_index_density システムプロシージャー, 1101
sa_table_fragmentation, 1197
断片化解除
REORGANIZE TABLE, 877

ち

チェックサム
VALIDATE CHECKSUM 文, 983

検証, 983
データベース設定の変更, 410
データベースの起動, 936
データベースの作成, 523
チェックポイント
CHECKPOINT 文の使用, 509
CHECKPOINT 文を使用したデータベースのチェックポイント, 509
チェックポイントログ
CHECKPOINT 文, 509
置換文字
CHAR と NCHAR の比較, 129
損失を伴う変換, 128
抽出テーブル
サブクエリの使用, 682
調整
ALTER DATABASE 文を使用したデータベースサーバーの調整, 408
ALTER DATABASE 文を使用してコストモデルを調整, 408
sa_load_cost_model システムプロシージャーを使用したコストモデルのロードとアンロード, 1109
sa_unload_cost_model システムプロシージャーを使用したコストモデルのロードとアンロード, 1206
並列 I/O 機能, 411
直列化可能
FROM 句, 764
著作権
取り出し, 1238

つ

追加
ALTER TABLE 文を使用してカラムを追加, 462
CREATE INDEX 文を使用してインデックスを追加, 567
Java クラス, 809
サーバー, 618
メッセージ, 578
通貨
通貨データ型, 103
通貨データ型
MONEY, 103
SMALLMONEY, 103
通貨データ型のアルファベット順リスト
説明, 103
通信プロトコル
Mobile Link の複数設定, 650
通知
エラー, 862
例外, 879

て

底が 10 の対数
LOG10 関数, 273
定義
ALTER TABLE 文を使用してテーブルを変更, 462
停止
Interactive SQL のロギング, 949
Java VM, 948
STOP EXTERNAL ENVIRONMENT 文を使用した外部環境の停止, 947
データベース, 946
データベースサーバー, 949
パススルーモード, 855
定数
Transact-SQL, 33
構文, 14
説明, 6
ディスク転送時間モデル
ALTER DATABASE 文を使用した調整, 408
ALTER DATABASE 文を使用してデフォルトをリストア, 408
現在の値, 1088
ディスク領域
CREATE EVENT 文を使用してイベントを作成, 540
空き領域の決定, 1080
ディスク領域不足イベントの作成, 540
ディスク領域不足
CREATE EVENT 文を使用してイベントを作成, 540
ディレクトリアクセスサーバー
CREATE SERVER 文, 618
適合度
回帰直線, 321
テキスト
READTEXT 文を使用してデータベースから読み込み, 866
テキストインデックス
ALTER TEXT INDEX 文, 480

COMMENT 文を使用してコメントを追加, 512
CREATE TEXT INDEX 文, 667
REFRESH TEXT INDEX を使用したリフレッシュ, 870
sa_refresh_text_indexes を使用した再表示, 1158
sa_text_index_stats を使用したリスト化, 1201
SYSMVOPTION に格納されている作成オプション, 1282
検証, 983
削除, 733
トランケート, 963
テキスト関数
説明, 151
テキスト設定オブジェクト
ALTER TEXT CONFIGURATION 文, 476
COMMENT 文を使用してコメントを追加, 512
CREATE TEXT CONFIGURATION 文, 666
sa_char_terms での文字列を単語に分割する方法のテスト, 1044
sa_nchar_terms での文字列を単語に分割する方法のテスト, 1139
削除, 733
テストの種類
openxml システムプロシージャーがサポート, 1038
デッドロック
sa_report_deadlocks システムプロシージャー, 1160
ロギング, 1178
デッドロックレポート
sa_report_deadlocks システムプロシージャー, 1160
デバッグ
MESSAGE 文の動作の制御, 841
TRACEBACK, 372
デフォルト
CREATE TABLE 文, 656
INSERT 文, 802
デフォルト値
CURRENT_TIMESTAMP, 64
CURRENT_USER, 65
CURRENT DATABASE, 62
CURRENT DATE, 62
CURRENT PUBLISHER, 62
CURRENT REMOTE USER, 63
CURRENT TIME, 63
CURRENT TIMESTAMP, 64
CURRENT USER, 65
CURRENT UTC TIMESTAMP, 66
INSERT 文, 802
LAST USER, 67
SQLCODE, 70
SQLSTATE, 71
TIMESTAMP, 73
USER, 74
UTC TIMESTAMP, 75
オプションパラメーターに DEFAULT を使用, 503
電子メール
拡張システムプロシージャー, 1030
システムプロシージャー, 1241
テンポラリオプション
Interactive SQL の設定, 916
SET OPTION 文, 913
テンポラリ関数
説明, 563
テンポラリストアドプロシージャー
作成, 602
テンポラリテーブル
CREATE LOCAL TEMPORARY TABLE 文を使用してローカルテンポラリファイルを作成, 571
CREATE TABLE の使用, 662
CREATE TABLE 文, 651
Transact-SQL CREATE TABLE 文, 663
ローカルテンポラリテーブル上のビューの使用不可, 681
ローカルテンポラリテーブルの宣言, 691
テンポラリファイル
使用可能な領域の決定, 1080
テンポラリプロシージャー
CREATE PROCEDURE 文, 602
データ
テーブルからファイルへエクスポート, 848
ファイルからテーブルにインポート, 794
ローの選択, 895
データアクセス計画
テキスト仕様の取得, 747
データ型
ALTER DOMAIN 文を使用した変更, 416
BIGINT データ型, 94
BINARY データ型, 122
BIT データ型, 95
CHAR データ型, 85
CREATE DOMAIN 文, 533
DATETIMEOFFSET データ型, 114

DATETIME データ型, 113
DATE データ型, 112
DECIMAL データ型, 96
DOUBLE データ型, 97
DROP DATATYPE 文を使用してユーザー定義データ型を削除, 710
FLOAT データ型, 98
IMAGE データ型, 123
INTEGER データ型, 99
Java と SQL の変換, 136
LONG BINARY データ型, 123
LONG NVARCHAR データ型, 87
LONG VARBIT データ型, 104
LONG VARCHAR データ型, 88
NCHAR, 88
NCHAR (NATIONAL CHAR), 88
NTEXT データ型, 89
NUMERIC データ型, 99
NVARCHAR データ型, 89
REAL データ型, 101
SMALLDATETIME データ型, 116
SMALLINT データ型, 101
SMALLMONEY データ型, 103
SQL 変換関数, 141
SYSDOMAIN システムビュー, 1262
SYSEXTERNLOGIN システムビュー, 1267
SYSUSERTYPE システムビュー, 1330
TEXT データ型, 90
TIMESTAMP WITH TIME ZONE データ型, 120
TIMESTAMP データ型, 118
TIME データ型, 117
TINYINT データ型, 102
Unicode, 85
UNIQUEIDENTIFIERSTR データ型, 91
UNIQUEIDENTIFIER データ型, 123
VARBINARY データ型, 124
VARBIT データ型, 105
VARCHAR データ型, 91
XML データ型, 93
値の比較, 127
互換性, 132
取得, 229
特別値, 61
比較演算子についての変換, 127
丸め誤差, 93
文字, 85
ユーザー定義のドメイン, 125
データ型変換
CAST, 165
CHAR 値と NCHAR 値の比較, 129
DOUBLE を NUMERIC に変換する, 135
Java から SQL, 136
NCHAR を CHAR に変換する, 133
SQL から Java, 136, 137
式を評価する場合, 127
説明, 127
比較演算子, 127
データ型変換関数
説明, 141
データのアンロード
マルチバイト文字セット, 968
データのインポート
ファイルからテーブル, 794
データのエクスポート
テーブルからファイル, 848
データのエンコード
BASE64_ENCODE 関数, 159
HTML_ENCODE 関数, 245
データの復号化
BASE64_DECODE 関数, 158
HTML_DECODE 関数, 244
HTTP_DECODE 関数, 247
データのロード
マルチバイト文字セット, 823
データベース
ALTER DATABASE 文を使用して jConnect をアップグレード, 408
BACKUP 文を使用してバックアップ, 489
CHECKPOINT 文を使用したチェックポイント, 509
CONNECT 文を使用して接続, 516
CREATE DATABASE 文を使用して作成, 521
CREATE DBSPACE 文を使用してファイルを作成, 528
DROP DATABASE 文を使用してファイルを削除, 709
sa_validate システムプロシージャーでの検証, 1210
SYSFILE システムビュー, 1358
UNLOAD 文を使用したデータのアンロード, 966
アーカイブからリストア, 880
起動, 935
現在のデータベースの場所を返す, 206
システムテーブル, 993
システムプロシージャー, 1029

スキーマ (テーブルのリスト), 1253
スキーマ (ビューのリスト), 993
接続の無効化, 1170
デフォルトのシステムビュー, 1253
データベースの暗号化されたコピーの作成, 535
バルクデータのロード, 814
マイグレート, 1127
データベース ID 番号
DB_ID 関数, 207
データベースオブジェクト
COMMENT 文を使用してコメントを追加, 512
識別, 4
データベースオプション
quoted_identifier と T-SQL の互換性, 34
Transact-SQL 互換性, 1227
Transact-SQL の設定, 929
カーソルからローをフェッチ, 915
初期設定と sp_login_environment システムプロシージャー, 1217
初期設定と sp_tsql_environment システムプロシージャー, 1227
ユーザー定義のオプション, 914
データベースクリーナー
sa_clean_database システムプロシージャー, 1046
説明, 1046
データベース検証
VALIDATE CHECKSUM 文, 983
VALIDATE INDEX 文, 983
データベースサーバー
sa_server_option システムプロシージャーの設定オプション, 1170
START SERVER 文, 938
STOP SERVER 文, 949
データベースミラーリング、ALTER MIRROR SERVER 文, 430
データベースミラーリング、CREATE MIRROR SERVER 文, 579
読み込み専用のスケールアウト、ALTER MIRROR SERVER 文, 430
読み込み専用のスケールアウト、CREATE MIRROR SERVER 文, 579
データベースサーバーメッセージウィンドウ
メッセージの表示, 841
データベーススキーマ
システムテーブル, 993
システムビュー, 1253
データベース抽出
SQL Remote REMOTE RESET 文, 874
データベースのアップグレード
ALTER DATABASE 文, 408
データベースの暗号化
CREATE ENCRYPTED DATABASE 文, 535
データベースの作成
CREATE DATABASE 文, 521
データベースの停止
STOP DATABASE 文, 946
データベースのマイグレート
sa_migrate システムプロシージャー, 1127
データベースファイル
CREATE DECRYPTED DATABASE 文を使用した復号化, 530
CREATE DECRYPTED FILE 文を使用した復号化, 531
CREATE ENCRYPTED DATABASE 文を使用した暗号化, 535
CREATE ENCRYPTED FILE 文を使用した暗号化, 537
DROP DATABASE 文を使用して削除, 709
インデックスの格納, 569
データベースミラーリング
ALTER MIRROR SERVER 文, 430
CREATE MIRROR SERVER 文, 579
LOAD TABLE 文の制限, 826
SET MIRROR OPTION 文, 908
サーバーの削除, 719
フェールオーバーの起動, 410
データベース名
DB_NAME 関数によって返される, 208
テープドライブ
BACKUP 文を使用してデータベースバックアップを作成, 489
テーブル
ALTER TABLE 文, 462
ALTER TABLE 文を使用した変更, 462
COMMENT 文を使用してコメントを追加, 512
CREATE EXISTING TABLE 文を使用してプロキシテーブルを作成, 546
CREATE LOCAL TEMPORARY TABLE 文を使用してローカルテンポラリファイルを作成, 571
CREATE TABLE 文, 651
DROP TABLE 文を使用して削除, 731
sa_get_table_definition システムプロシージャーでの定義の生成, 1095

SQL Remote 更新, 973
UNLOAD 文を使用したデータのアンロード, 966
依存性の決定, 1070
再編成, 877
データをファイルへエクスポート, 848
トランケート, 961
名前の変更, 472
バルクロード, 814
ファイルからデータをインポート, 794
ロック, 831
ローカルテンポラリの作成, 691
ローを挿入, 802
テーブル制約
CREATE TABLE 文, 658
テーブルの暗号化
ALTER TABLE 文, 462
CREATE ENCRYPTED TABLE DATABASE 文, 535
CREATE ENCRYPTED TABLE DATABASE 文を使用した暗号化, 535
テーブルのインデックス
Interactive SQL での表示, 704
テーブルのカラム
Interactive SQL での表示, 704
テーブルの再編成
REORGANIZE TABLE, 877
テーブルの削除
DROP TABLE 文, 731
テーブルの作成
CREATE TABLE 文, 651
テーブルの制約
ALTER TABLE 文を使用した追加、削除、変更, 462
ALTER TABLE 文を使用して追加, 469
ALTER TABLE 文を使用して変更, 472
テーブルの復号化
ALTER TABLE 文, 462
テーブル番号
システムビュー, 1311
テーブルヒント
FROM 句, 764
テーブルページ
ALTER TABLE 文を使用して PCTFREE を設定, 462
CREATE LOCAL TEMPORARY TABLE 文を使用して PCTFREE を設定, 571
LOAD TABLE 文を使用した PCTFREE の設定, 814
PCTFREE の設定, 651
テーブルリスト
FROM 句, 761
テーブルロック
sa_locks システムプロシージャー, 1112

と

同期サブスクリプションの作成
CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link], 647
同期プロファイルの削除
DROP SYNCHRONIZATION PROFILE 文 [Mobile Link], 729
同期プロファイルの作成
CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link], 646
同期プロファイルの変更
ALTER SYNCHRONIZATION PROFILE 文 [Mobile Link], 457
統計
CREATE STATISTICS 文, 643
LOAD TABLE を使用して部分的に更新, 827
sa_get_histogram システムプロシージャーを使用して取得, 1089
SYSCOLSTAT システムビュー, 1258
ディスクへのフラッシュ, 1085
ロード, 814
統計クリーナー
sa_server_option での制御, 1184
統計情報
ALTER SERVICE 文を使用した更新, 456
DROP STATISTICS 文を使用して削除, 727
統合化ログイン
COMMENT 文を使用してコメントを追加, 512
GRANT 文, 783
REVOKE 文, 887
統合データベース
SQL Remote パーミッションの取り消し, 884
統合パーミッションの取り消し
REVOKE 文, 887
統合ビュー
SYSARTICLECOLS, 1340
SYSARTICLES, 1341
SYSCAPABILITIES, 1341
SYSCATALOG, 1342

SYSCOLAUTH, 1342
SYSCOLSTATS, 1343
SYSCOLUMNS, 1343
SYSGROUPS, 1345
SYSINDEXES, 1345
SYSOPTIONS, 1346
SYSPROCAUTH, 1346
SYSPROCPARMS, 1347
SYSPROCS, 1347
SYSPUBLICATIONS, 1348
SYSREMOTEOPTION2, 1348
SYSREMOTEOPTIONS, 1349
SYSREMOTETYPES, 1349
SYSREMOTEUSERS, 1350
SYSSUBSCRIPTIONS, 1350
SYSSYNC2, 1351
SYSSYNCPUBLICATIONDEFAULTS, 1351
SYSSYNCS, 1352
SYSSYNCSCRIPTS, 1352
SYSSYNCSUBSCRIPTIONS, 1353
SYSSYNCUSERS, 1353
SYSTABAUTH, 1354
SYSTRIGGERS, 1354
SYSUSERAUTH, 1362
SYSUSEROPTIONS, 1355
SYSVIEWS, 1356
説明, 1335
同時実行性
テーブルのロック, 831
動的 SQL
プロシージャーの実行, 740
特殊なテーブル
説明, 993
特殊なビュー
説明, 1253
特殊文字
SQL 文字列, 7
全文クエリ文字列で使用できる構文, 55
バイナリで使用, 6
文字列で使用, 7
特殊文字クラス
正規表現, 22
特別値
CREATE TABLE 文, 465, 656
CURRENT_TIMESTAMP, 64
CURRENT_USER, 65
CURRENT DATABASE, 62
CURRENT DATE, 62
CURRENT PUBLISHER, 62
CURRENT REMOTE USER, 63
CURRENT TIME, 63
CURRENT TIMESTAMP, 64
CURRENT USER, 65
CURRENT UTC TIMESTAMP, 66
LAST USER, 67
NULL, 67
SQLCODE, 70
SQLSTATE, 71
TIMESTAMP, 73
USER, 74
UTC TIMESTAMP, 75
説明, 61
独立性レベル
ISOLATION LEVEL 句, 845
句, 845
テーブルヒント, 764
独立変数
回帰直線, 316
閉じる
CLOSE 文を使用してカーソルを閉じる [ESQL] [SP], 510
DROP CONNECTION 文を使用して接続を閉じる, 709
Interactive SQL, 746
ドメイン
ALTER DOMAIN 文を使用した変更, 416
CREATE DOMAIN 文, 533
DROP DOMAIN 文を使用して削除, 712
NULL 入力属性, 533
Transact-SQL, 126
説明, 125
ドメインの作成
CREATE DOMAIN 文, 533
トラップ
Embedded SQL 内のエラー, 987
トラブルシューティング
標準的でないディスクドライブ, 411
ログ操作, 1179
ロック, 1111
トランケート
テキストインデックス, 963
テーブル, 961
テーブルが即時ビューから参照されていると失敗, 961
テーブルに即時テキストインデックスがあると失敗, 961

トランザクション
BEGIN TRANSACTION 文を使用してユーザー定義トランザクションをネスト, 500
BEGIN TRANSACTION 文を使用してユーザー定義を開始, 500
COMMIT 文を使用してコミット, 514
ROLLBACK TO SAVEPOINT 文を使用してセーブポイントへロールバック, 891
ROLLBACK 文を使用したロールバック, 890
セーブポイントの作成, 895
ロールバック, 892
ロールバックと SAVE TRANSACTION 文, 894
トランザクション管理
BEGIN TRANSACTION 文, 500
Transact-SQL, 514
Transact-SQL 内, 500
トランザクションの独立性レベルオプション
Transact-SQL SET 文を使用して設定, 929
トランザクションモード
非連鎖, 501
連鎖, 501
トランザクションログ
ALTER DBSPACE 文を使用して領域を割り付け, 414
BACKUP 文を使用してバックアップ, 489
CREATE DECRYPTED DATABASE 文を使用した復号化, 530
CREATE DECRYPTED FILE 文を使用した復号化, 531
CREATE ENCRYPTED DATABASE 文を使用した暗号化, 535, 537
TRUNCATE TABLE 文, 961
使用可能な領域の決定, 1080
バックアップなしでトランケート, 490
バックアップなしで名前を変更, 490
トランザクションログミラー
CREATE DECRYPTED DATABASE 文を使用した復号化, 530
CREATE DECRYPTED FILE 文を使用した復号化, 531
CREATE ENCRYPTED DATABASE 文を使用した暗号化, 535, 537
使用可能な領域の決定, 1080
トリガー
@@identity グローバル変数, 83
ALTER TRIGGER 文を使用して変更, 481
COMMENT 文を使用してコメントを追加, 512
CREATE TRIGGER 文を使用して生成, 670
DELETING 条件, 58
DROP FUNCTION 文を使用して削除, 714
DROP 文を使用して削除, 734
INSERTING 条件, 58
Transact-SQL の作成, 676
TRUNCATE TABLE 文, 962
UPDATING 条件, 58
イベント, 960
オペレーション条件, 58
ロールバック, 893
ローレベル, 673
取り消し
REVOKE 文, 887
SQL Remote CONSOLIDATE パーミッション, 884
SQL Remote PUBLISH パーミッション, 885
SQL Remote REMOTE DBA パーミッション, 886
SQL Remote REMOTE パーミッション, 887
トランザクションのロールバックによる変更, 890
トレーシング
ATTACH TRACING 文, 487
DETACH TRACING 文, 706
REFRESH TRACING LEVEL 文, 872
トレーシングデータ
sa_save_trace_data システムプロシージャーを使用した保存, 1165
トレーシングレベル
sa_set_tracing_level システムプロシージャーを使用した設定, 1191

な

内部格納フォーマット
ALTER SPATIAL REFERENCE SYSTEM 文, 455
CREATE SPATIAL REFERENCE SYSTEM 文, 640
長いカラム名
取得, 702
名前
カラム名, 14
名前付きパラメーター
プロシージャーを呼び出すときに許可される, 503
名前の変更
カラム, 472

制約, 472
テーブル, 472
名前変更
ALTER TABLE 文を使用したカラムの名前変更, 462
ALTER TABLE 文を使用したテーブルの名前変更, 462

に

二重引用符
SQL 識別子で使用できない, 4
データベースオブジェクト, 4
二重スラッシュ
コメントインジケーター, 84
二重ハイフン
コメントインジケーター, 84

ね

ネイティブ関数呼び出し
CREATE FUNCTION 文を使用したネイティブ関数へのインターフェイス, 550
CREATE PROCEDURE 文を使用したネイティブプロシージャーへのインターフェイス, 583
関数, 551
プロシージャー, 587
ネイティブプロシージャー
CREATE PROCEDURE 文を使用したネイティブプロシージャーへのインターフェイス, 583
ネスト
BEGIN TRANSACTION 文を使用してユーザー定義トランザクションをネスト, 500
年内の日の番号
ISO 8601, 107

は

排他 OR
ビット処理演算子, 11
バイトオーダーマーク
UTF-16 または UTF-8 データファイルからデータをロード, 826
バイナリ
エスケープ文字, 6
バイナリデータ型
BINARY, 122
IMAGE, 123
LONG BINARY, 123
UNIQUEIDENTIFIER, 123
VARBINARY, 124
エンコード, 159
カラムから取得, 771
復号化, 158
バイナリデータ型のアルファベット順リスト
説明, 122
バイナリラージオブジェクト
ASE 生成の BCP ファイルのインポート, 821
GET DATA 文, 771
SET 文の例, 928
xp_read_file システムプロシージャーを使用して挿入, 1239
エクスポート, 1250
カラムから取得, 771
トランザクションログの考慮事項, 414
バイナリデータ型, 122
バイナリリテラル
特殊文字, 6
ハイフン
CONTAINS 句で使用できる構文, 55
全文クエリ文字列で使用できる構文, 55
バインド変数
EXECUTE 文, 743
OPEN 文, 845
カーソルの記述, 700
パススルーモード
PASSTHROUGH 文 [SQL Remote], 855
起動, 855
停止, 855
パスワード
sa_verify_password システムプロシージャー, 1212
最大長、ALTER USER 文, 483
最大長、CREATE USER 文, 678
最大長、GRANT 文, 785
文字セット変換、CREATE USER 文, 678
文字セット変換、GRANT 文, 785
パスワードの文字セット変換
ALTER USER 文, 483
パスワード、文字セット変換
ALTER USER 文, 483
派生テーブル
FROM 句 SQL 構文, 758
FROM 句での例, 761
ラテラル, 762
パターン一致
LIKE 探索条件, 42
PATINDEX 関数, 297

REGEXP 探索条件, 47
SIMILAR TO 探索条件, 48
大文字と小文字の区別, 43
照合, 43
パターン長, 43
パターン長
LIKE 探索条件, 43
バックアップ
BACKUP 権限, 783
BACKUP 文, 489
BACKUP 文を使用して作成, 489
BACKUP 文を使用してテープにバックアップ, 489
CREATE EVENT 文を使用してイベントを作成, 540
データベースのリストア, 880
読み込み専用データベースサーバーでの開始, 493
パッケージ
Java クラスのインストール, 809
Java クラスの削除, 876
ハッシュ処理
サポートされるアルゴリズム, 239
発生
RAISERROR 文, 862
パフォーマンス
I/O コストモデルの再調整, 411
SQL Remote 更新, 975
圧縮統計, 1052
データベースサーバーの再調整, 408
領域の事前割り付け, 414
パフォーマンスモニター
実行時間の判別, 1092
パブリケーション
ALTER PUBLICATION 文, 435
CREATE PUBLICATION 文, 610
DROP PUBLICATION 文, 721
SQL Remote UPDATE 文, 975
UPDATE 文, 980
パブリッシャー
GRANT PUBLISH 文, 777
REMOTE, 780
SQL Remote アドレス, 436, 613
アドレス, 722
パラメーター
Interactive SQL スクリプトファイル, 854
パラメーター化されたビュー
説明, 682
バルクオペレーション
UNLOAD 文を使用したデータのアンロード, 966
バルクロード
LOAD TABLE 文, 814
範囲
データ型, 112
反復
カーソル, 753
バージョン番号
取り出し, 1238
パーセント記号
コメントインジケーター, 84
パートナーサーバー
ALTER MIRROR SERVER 文, 430
ALTER MIRROR SERVER 文を使用した変更, 431
CREATE MIRROR SERVER 文, 579
CREATE MIRROR SERVER 文を使用した定義, 580
パーミッション
ALL の取り消し, 887
ALTER の取り消し, 887
BACKUP の取り消し, 887
CONNECT の取り消し, 887
CONSOLIDATE の付与, 776
CREATE ON の取り消し, 887
DBA の取り消し, 887
DB 領域, 783
DELETE の取り消し, 887
EXECUTE の取り消し, 887
GRANT 文, 782
GROUP の取り消し, 887
INSERT の取り消し, 887
INTEGRATED LOGIN の取り消し, 887
KERBEROS LOGIN の取り消し, 887
MEMBERSHIP IN GROUP の取り消し, 887
PROFILE の取り消し, 887
PUBLISH の付与, 777
REFERENCES の取り消し, 887
REMOTE の付与, 780
RESOURCE の取り消し, 887
SELECT の取り消し, 887
SQL Remote CONSOLIDATE の取り消し, 884
SQL Remote PUBLISH の取り消し, 885
SQL Remote REMOTE DBA の取り消し, 886
SQL Remote REVOKE REMOTE 文, 887
SYSCOLAUTH ビュー, 1342

SYSCOLPERM システムビュー, 1257
SYSTABAUTH 統合ビュー, 1354
SYSTABLEPERM システムビュー, 1318
UPDATE の取り消し, 887
VALIDATE の取り消し, 887
シーケンス, 783
シーケンスに関する取り消し, 888
取り消し, 887
プロシージャー, 782
無効化されたオブジェクト, 786

ひ

比較
CHAR 値と NCHAR 値, 129
CHAR と NCHAR, 129
COMPARE 関数, 171
時刻, 130
数値データ型, 130
探索条件, 34
日付, 130
比較演算子
Transact-SQL との互換性, 8
構文, 8
データ変換, 127
比較演算子を使用した変換
説明, 127
ヒストグラム
CREATE STATISTICS を使用した更新, 643
CREATE STATISTICS を使用した作成, 643
LOAD TABLE を使用して部分的に更新, 827
SYSCOLSTAT システムビュー, 1258
取得, 1089
選択性推定, 60
日付
2 月 29 日, 111
SQL Anywhere, 105
閏年, 111
解釈, 112
格納, 106
クエリ, 111
現在のシステム日付のクエリ, 236
取得, 112
挿入, 112
データベースへの送信, 106
テーブルの生成, 1165
比較, 130
変換関数, 142
文字列からの変換, 132
日付関数
アルファベット順リスト, 142
日付と時刻データ型
DATE, 112
DATETIME, 113
DATETIMEOFFSET, 114
ISO 8601, 106
SMALLDATETIME, 116
TIME, 117
TIMESTAMP, 118
TIMESTAMP WITH TIME ZONE, 120
説明, 105
日付と時刻データ型のアルファベット順リスト
説明, 105
日付と時刻のデータベースからの取得
説明, 111
日付と時刻の比較
説明, 130
日付と時刻のデータベースに送信する
説明, 106
日付の単位
説明, 142
ビット
変換, 134
ビット処理演算子
構文, 11
ビット配列
説明, 104
データ型, 104
変換, 134
ビット配列データ型
LONG VARBIT, 104
VARBIT, 105
説明, 104
ビット配列データ型のアルファベット順リスト
説明, 104
ビット配列の変換
説明, 134
ビュー
ALTER VIEW 文を使用して変更, 484
COMMENT 文を使用してコメントを追加, 512
CREATE MATERIALIZED VIEW 文, 576
CREATE VIEW 文, 681
DROP VIEW 文, 737
INSERT 文を使用した更新, 805
sa_recompile_views システムプロシージャー, 1157

Transact-SQL 互換性, 1365
依存性の決定, 1070
インデックス, 570
互換ビュー, 1356
システムビュー, 1254
説明, 1253
統合ビュー, 1335
パラメーター化されたビュー, 682
別のユーザーが所有するマテリアライズド
ビューの変更, 485
ビューの依存性
データベースのアンロード／再ロード, 1157
ビューの削除
DROP VIEW 文, 737
ビューの作成
CREATE VIEW 文, 681
表示
Interactive SQL プロシージャープロファイリング
グデータ, 1154
メッセージ, 841
メッセージウィンドウにメッセージを表示,
859
標準偏差
STDDEV_POP 関数, 354
STDDEV_SAMP 関数, 356
STDDEV 関数, 354
標本共分散
説明, 188
ヒント
FROM 句を使用したインデックスヒント, 766
頻度
メッセージの送信、GRANT CONSOLIDATE
文 [SQL Remote], 776
メッセージの送信、GRANT REMOTE 文 [SQL
Remote], 780

ふ

ファイル
CREATE DBSPACE 文を使用してデータベー
スを作成, 528
CREATE DECRYPTED DATABASE 文を使用
した復号化, 530
CREATE DECRYPTED FILE 文を使用した復号
化, 531
CREATE ENCRYPTED FILE 文を使用して暗号
化, 537
DB 領域の割り付け, 414
SQL 文の読み込み, 863
xp_read_file システムプロシージャー, 1239
xp_write_file システムプロシージャー, 1250
クライアントコンピューターでの読み込み,
313
クライアントコンピューターへの書き込み,
390
データをテーブルにインポート, 794
テーブルからデータをエクスポート, 848
ファイル内のクエリ, 762
ファイルからの SQL 文の読み込み
説明, 863
ファイルサイズ
CREATE EVENT 文を使用してイベントを作
成, 540
ファイルの書き込み
xp_write_file の使用, 1250
ファイルの読み込み
xp_read_file の使用, 1239
フェッチ
ローをカーソルからフェッチ, 749
フォレスト
解析対象外の XML ドキュメント内, 397
復号化
ALTER TABLE 文を使用してテーブルを復号,
462
CREATE DECRYPTED DATABASE 文を使用
したファイル, 530
CREATE DECRYPTED FILE 文を使用したファ
イル, 531
SQL を使用してマテリアライズドビューを復
号化, 427
複合文
BEGIN 文, 497
TSQL 互換性, 499
使用 [TSQL], 499
複数の結果セット
取得, 881
プット
カーソルへローをプット, 860
物理インデックス
SYSPHYSIDX システムビューに記録, 1285
部分文字クラス
REGEXP 探索条件, 47
SIMILAR TO 探索条件, 48
正規表現, 22
部分文字列
説明, 361

置換, 330
付与
CONSOLIDATE パーミッション, 776
PUBLISH パーミッション, 777
REMOTE DBA 権限, 778
REMOTE パーミッション, 780
権限, 782
パーミッション, 782
プライマリキー
ALTER INDEX 文, 422
ALTER INDEX 文を使用したクラスタリング, 422
ALTER INDEX 文を使用した名前変更, 422
CREATE SEQUENCE 文, 616
CREATE TABLE 文でのカラムの順序, 659
CREATE TABLE 文での整合性の制約, 659
UUID と GUID, 286
UUID を使用したユニークな値の生成, 286
ユニークな値の生成, 286
リモートテーブルと sp_remote_exported_keys システムプロシージャー, 1219
リモートテーブルと sp_remote_imported_keys システムプロシージャー, 1221
プライマリキーロック
sa_locks システムプロシージャー, 1112
プライマリサーバー
ALTER MIRROR SERVER 文を使用した変更, 430
CREATE MIRROR SERVER 文を使用した定義, 580
プライマリテーブル
システムビュー, 1268
プラン
EXPLANATION 関数, 228
GRAPHICAL_PLAN 関数, 236
PLAN 関数, 302
TRACED_PLAN 関数, 373
カーソル, 302
プランをファイルに保存する例, 237
プレフィックス検索
CONTAINS 探索条件, 51
プロキシテーブル
CREATE EXISTING TABLE 文を使用して作成, 546
CREATE TABLE 文, 653
プロキシプロシージャー
Web サービスの作成, 591
プロシージャー
ALTER PROCEDURE 文を使用した変更, 433
ALTER PROCEDURE 文を使用したレプリケーション, 433
CALL 文を使用して起動, 503
COMMENT 文を使用してコメントを追加, 512
CREATE PROCEDURE 文, 599
CREATE PROCEDURE 文を使用したネイティブプロシージャーへのインターフェイス, 583
DROP PROCEDURE 文を使用して削除, 720
GRANT 文, 782
RAISERROR 文, 862
SQL ストアドプロシージャーの作成, 602
Transact-SQL ストアドプロシージャーの実行, 744
Transact-SQL の作成, 599
Transact-SQL のリスト, 1033
Web サービスの置き換え, 592
Web サービスの作成, 591
値を返す, 882
アルファベット順リスト, 1035
外部プロシージャーインターフェイスの置き換え, 585
拡張リスト, 1030
関数との比較, 145
システム, 1029
システムプロシージャーのアルファベット順リスト, 1035
実行の再開, 881
終了, 882
選択, 761
動的 SQL の実行, 740
ネイティブ関数呼び出し, 587
変数結果セット、CREATE PROCEDURE 文 [外部呼び出し], 585
変数結果セット、CREATE PROCEDURE 文, 604
変数結果セット、DESCRIBE 文 [ESQL], 702
変数結果セット PREPARE 文, 857
ユーザー定義プロシージャーの置き換え, 603
呼び出すときに許可される名前付き, 503
プロシージャーコール
CALL 文を使用して起動, 503
プロシージャーのプロファイリング
sa_procedure_profile システムプロシージャー, 1152
プロシージャーの呼び出し
CALL 文, 503
プロシージャーパラメーター

Interactive SQL での表示, 704
プロシージャープロファイリング
Interactive SQL, 1170
Interactive SQL からの無効化, 1180
Interactive SQL からの有効化, 1180
Interactive SQL での表示, 1154
プロシージャーの概要, 1154
ブロック
識別, 1055
トラブルシューティング, 1111
ブロックフェッチ
FETCH 文, 750
OPEN 文, 846
プロパティ
CONNECTION_PROPERTY 関数, 178
DB_PROPERTY 関数, 208
PROPERTY 関数, 304
プロファイリング
PROFILE 権限, 783
文
BEGIN 文にグループ化, 497
BEGIN 文にグループ化 [TSQL], 499
SQL Anywhere サーバーの文のアルファベット順リスト, 406
準備, 856
準備文の削除, 726
準備文の実行, 742
分岐
MERGE 文, 839
分岐化
MERGE 文, 839
文の構文
SQL Anywhere サーバーの文のアルファベット順リスト, 406
マニュアルの表記規則, 403
文ラベル
GOTO Transact-SQL 文, 775
プール
接続プールの有効化, 932

へ

平均関数
AVG 関数, 156
平方根関数
SQRT 関数, 353
平面モデル
ALTER SPATIAL REFERENCE SYSTEM 文, 454
CREATE SPATIAL REFERENCE SYSTEM 文, 637
並列バックアップ
BACKUP 文, 489
変換
CAST, 165
DOUBLE を NUMERIC に変換する, 135
NCHAR から CHAR, 133
SQL と Java, 136
あいまいな日付と文字列, 108
式を評価する場合, 127
データ型変換, 127
比較演算子の使用, 127
ビット, 134
ビット配列, 134
文字列から日付, 132
変換関数
アルファベット順リスト, 141
データ型, 141
変換文字列
説明, 148
変更
ALTER DATABASE 文を使用してデータベースを変更, 408
ALTER DBSPACE 文を使用して DB 領域を変更, 414
ALTER DOMAIN 文を使用してデータ型を変更, 416
ALTER DOMAIN 文を使用してドメインを変更, 416
ALTER EVENT 文を使用してイベントを変更, 417
ALTER INDEX 文を使用してインデックスを変更, 422
ALTER LOGIN POLICY 文を使用してログインポリシーオプションを変更, 424
ALTER MATERIALIZED VIEW 文を使用してマテリアライズドビューを変更, 427
ALTER PROCEDURE 文を使用してプロシージャーを変更, 433
ALTER PUBLICATION 文, 435
ALTER SERVER 文を使用してリモートサーバー属性を変更, 439
ALTER TABLE 文, 462
ALTER TABLE 文を使用してカラムを変更, 462

ALTER TEXT INDEX 文を使用してテキストインデックスを変更, 480
ALTER TRIGGER 文を使用してトリガーを変更, 481
ALTER USER 文を使用してログインポリシーオプションを変更, 482
ALTER VIEW 文を使用してビューを変更, 484
SQL Remote ALTER PUBLICATION 文, 435
SQL Remote リモートメッセージタイプ, 436
テキスト設定オブジェクト, 476
データ型, 127
変数
DECLARE 文を使用した宣言, 693
DROP VARIABLE 文を使用して SQL 変数を削除, 736
SQL の作成, 679
値の設定, 927
記述子領域内から取得, 773
グローバル変数, 79
構文, 76
初期値, 693
接続レベル変数, 78
ビュー定義での使用, 682
ローカル変数, 77
変数結果セット
プロシージャーから、CREATE PROCEDURE 文 [外部呼び出し], 585
プロシージャーから、CREATE PROCEDURE 文, 604
プロシージャーから、DESCRIBE 文 [ESQL], 702
プロシージャーから、PREPARE 文, 857
返送
例外, 879
ページサイズ
データベースの作成, 526
ページの使用状況
テーブル, 1198
ベーステーブル
CREATE TABLE 文, 662

ほ

母共分散
説明, 187
保護された機能
sa_server_option を使用した変更, 1183
ホスト変数
Embedded SQL の宣言, 685
一般的な SQL 構文要素, 403
母分散
説明, 384
ポリシーオプション
ALTER LOGIN POLICY 文を使用してポリシーオプションを変更, 424
ALTER USER 文を使用して変更, 482

ま

マテリアライズドビュー
ALTER INDEX 文, 422
ALTER MATERIALIZED VIEW 文, 427
COMMENT 文を使用してコメントを追加, 512
CREATE MATERIALIZED VIEW 文, 576
DROP MATERIALIZED VIEW 文, 717
REFRESH MATERIALIZED VIEW 文, 867
SQL を使用した最適化での使用の無効化, 427
SQL を使用した最適化での使用の有効化, 427
SQL を使用して暗号化, 427
SQL を使用して削除, 717
SQL を使用して作成, 576
SQL を使用して復号化, 427
SQL を使用して変更, 427
SQL を使用して無効化, 427
SQL を使用して有効化, 427
SQL を使用してリフレッシュ, 867
SYSMVOPTION に格納されている作成オプション, 1282
UNLOAD 文を使用したデータのアンロード, 966
インデックスの検証, 983
インデックスヒントを使用したクエリ, 766
ステータスの決定, 1118
即時ビューにするための適格性のテスト, 1116
即時ビューに必要なパーミッション, 428
データベースのすべてのマテリアライズドビューをリスト, 1118
テーブルヒントを使用したクエリ, 764
別のユーザーが所有するマテリアライズドビューの変更, 428
リフレッシュの独立性レベルの設定, 867
マテリアライズドビューの削除
DROP MATERIALIZED VIEW 文, 717
マテリアライズドビューの作成
CREATE MATERIALIZED VIEW 文, 576
マテリアライズドビューのプロパティ

RefreshType プロパティ, 1119
マテリアライズドビューのリフレッシュ
REFRESH MATERIALIZED VIEW 文, 867
REFRESH 句、ALTER MATERIALIZED VIEW 文, 428
マニュアル
SQL 構文の表記規則, 403
マルチバイト文字セット
データのアンロード, 968
データのロード, 823
マルチプログラミングレベル
現在の値の設定, 1178
最小値の設定, 1179
最大値の設定, 1179
自動チューニングの制御, 1177
統計の表示, 1177
マルチロー挿入
説明, 743
マルチローフェッチ
FETCH 文, 750
OPEN 文, 846
丸め誤差
DOUBLE, 97
FLOAT, 98
REAL, 101
説明, 93
マージ
MERGE 文, 834

み

未加工プロパティ
値の設定, 1209
値の増分, 1208
ミラーサーバー
ALTER MIRROR SERVER 文, 430
ALTER MIRROR SERVER 文を使用した変更, 430
COMMENT 文を使用してコメントを追加, 512
CREATE MIRROR SERVER 文, 579
CREATE MIRROR SERVER 文を使用した作成, 579
CREATE MIRROR SERVER 文を使用した定義, 580
DROP MIRROR SERVER 文での削除, 719
ミラーリング
サーバーの削除, 719

む

無効化されたオブジェクト
パーミッション, 786

め

明示的な選択性推定
説明, 60
メタプロパティ
LOAD TABLE 文, 822
openxml システムプロシージャー, 1036
メタ文字
正規表現に使用されるメタ文字のリスト, 18
メッセージ
DROP MESSAGE 文を使用してメッセージを削除, 718
MESSAGE 文, 841
SQL Remote のリモートタイプの作成, 613
SQL Remote のリモートタイプの変更, 436
作成, 578
表示, 841
リモートタイプの削除, 722
[メッセージ] ウィンドウ
メッセージを出力, 859
メッセージ制御パラメーター
設定, 919
メモリ
記述子領域への割り付け, 406

も

文字関数
アルファベット順リスト, 148
文字クラス
特殊文字クラスのサポート, 22
部分文字クラス, 22
文字セット
COMPARE 関数, 171
SORTKEY 関数, 347
式の評価時に変換する, 129
損失を伴う文字セット変換, 128
置換文字, 128
文字セット間の変換
説明, 129
文字セット変換
CHAR と NCHAR の比較, 129
NCHAR から CHAR への変換, 133
数値データ型間の比較, 130
損失を伴う変換, 128

置換文字, 128
データ型間の比較, 127
パスワード、CREATE USER 文, 678
パスワード、GRANT 文, 785
文字長のセマンティック
CHAR データ型, 85
VARCHAR データ型, 91
文字データ
格納, 85
文字列, 5
文字データ型
CHAR, 85
LONG NVARCHAR, 87
LONG VARCHAR, 88
NCHAR, 88
NTEXT, 89
NVARCHAR, 89
TEXT, 90
UNIQUEIDENTIFIERSTR, 91
VARCHAR, 91
XML, 93
説明, 85
文字の置換
CHAR と NCHAR の比較, 129
損失を伴う文字セット変換, 128
文字列
2 桁年のあいまいな変換, 108
SQL 関数, 148
Transact-SQL, 33
引用符, 33
エスケープ文字, 7
区切り文字列の解釈の変更, 34
後続ブランクの削除, 338
説明, 5
置換, 330
デリミター, 33
日付への変換, 132
文字列演算子
構文, 11
文字列関数
アルファベット順リスト, 148
文字列の位置
LOCATION 関数, 271
文字列の長さ
LENGTH 関数, 266
文字列リテラル
エスケープシーケンス, 7
説明, 7
特殊文字, 7
元の格納フォーマット
ALTER SPATIAL REFERENCE SYSTEM 文, 455
CREATE SPATIAL REFERENCE SYSTEM 文, 640
役割名
CREATE TABLE 文での外部キー, 660

ゆ

優先度
SQL 演算子の優先度, 12
ユニバーサルユニーク識別子
NEWID 関数の SQL 構文, 286
ユニークインデックス
説明, 568
ユーザー
ALTER SYNCHRONIZATION USER 文, 461
ALTER USER 文を使用して変更, 482
COMMENT 文を使用してコメントを追加, 512
CREATE SYNCHRONIZATION USER 文, 650
CREATE USER 文を使用して作成, 677
DROP SYNCHRONIZATION USER 文, 731
DROP USER 文を使用して削除, 735
削除, 887
ステータスの取得, 1096
設定, 932
ユーザー ID
システムビュー, 1311
制限, 785
取り消し, 887
ビュー, 1362
ユーザーが提供する選択性推定
説明, 60
ユーザー推定
説明, 60
ユーザー定義関数
CREATE FUNCTION 文, 563
Java, 144
値を返す, 882
終了, 882
説明, 144
定義済み, 144
ユーザー定義データ型
CREATE DOMAIN 文, 533
DROP DATATYPE 文を使用して削除, 710
Transact-SQL, 126

説明, 125
ユーザー定義のオプション
サポートされていないテンポラリ設定, 914
ユーザー定義プロパティ
値の設定, 1209
値の増分, 1208

よ

要求
タイミング情報の取得, 1140
要求のタイミング
sa_performance_diagnostics システムプロシージャー, 1140
要求のロギング
Interactive SQL での有効化, 1182
sa_get_request_profile での要求ログの分析, 1091
sa_get_request_times での要求ログの分析, 1092
要求ログ
sa_get_request_times プロファイルプロシージャーでの処理, 1181
要素
SQL 言語構文, 1
曜日
DOW 関数, 215
読み込み
データベースからテキスト値とイメージ値, 866
読み込み専用
テーブルのロック, 831
読み込み専用のミラーサーバーの定義, 580
リカバリ不要のバックアップ, 493
読み込み専用のスケールアウト
ALTER MIRROR SERVER 文, 430
CREATE MIRROR SERVER 文, 579
SET MIRROR OPTION 文, 908
サーバーの削除, 719
予約語
sa_reserved_words システムプロシージャー, 1161
SQL Anywhere サーバー, 1
SQL キーワードのリスト, 1
構文での使用方法, 1
識別子として使用, 34

ら

ラテラル派生テーブル
FROM 句の外部参照, 762
ラベル
文, 404, 775
ランキング関数
CUME_DIST 関数, 191
DENSE_RANK 関数, 213
PERCENT_RANK 関数, 300
RANK 関数, 311
アルファベット順リスト, 141
乱数
RAND 関数, 310
ラージデータベース
インデックス記憶領域, 569
ラージバイナリオブジェクト
カラムから取得, 771

り

リスト
LIST 関数の構文, 268
sa_split_list システムプロシージャー, 1194
リストア
アーカイブからデータベース, 880
リターンコード
EXIT 文 [Interactive SQL], 746
MAPI システムプロシージャーと SMTP システムプロシージャー, 1031
リテラル
説明, 6
リフレッシュ
テキストインデックス REFRESH TEXT INDEX, 870
リフレッシュタイプ
RefreshType プロパティ, 1119
リモートオプション
SET REMOTE OPTION 文 [SQL Remote], 919
リモートサーバー
ALTER SERVER 文を使用して属性を変更, 439
CREATE SERVER 文, 618
CREATE TABLE 文, 651
DROP SERVER 文を使用して削除, 723
SQL 文の送信, 757
SYSCAPABILITYNAME システムビュー, 1256
機能、sp_servercaps での確認, 1226
機能、SYSCAPABILITY システムビュー, 1255
切断, 439
リモートサーバーのログインの削除, 713
ログインの割り当て, 548

リモートデータアクセス
FORWARD TO 文, 757
切断, 439
リモートテーブル
CREATE TABLE 文, 653
sp_remote_tables システムプロシージャーでのリストの表示, 1224
外部キーと sp_remote_exported_keys システムプロシージャー, 1219
外部キーと sp_remote_imported_keys システムプロシージャー, 1221
カラム, 1218
プライマリキーと sp_remote_exported_keys システムプロシージャー, 1219
プライマリキーと sp_remote_imported_keys システムプロシージャー, 1221
リモートプロシージャー
CREATE PROCEDURE 文, 602
RESULT 句の要件, 606
Transact-SQL の作成, 599
リモートプロシージャーの削除
DROP PROCEDURE 文, 720
リモートメッセージタイプ
SQL Remote の作成, 613
SQL Remote の変更, 436
削除, 722
リモートユーザー
SQL Remote REVOKE REMOTE 文, 887
量指定子
式の量指定子, 18

る

ループ
カーソル, 753
ルール
SQL 言語構文, 1

れ

例外
DECLARE 文を使用した宣言, 693
DECLARE 文を使用した宣言, 693
通知, 934
返送, 879
暦日
ISO 8601, 107
レプリケーション
ALTER TABLE 文, 462
連結文字列
文字列演算子, 11

ろ

ロギング
START LOGGING 文, 940
STOP LOGGING 文, 949
デッドロック, 1178
ロギングなしでカラムを更新, 991
ログイン
サーバーへの接続の無効化, 1170
ステータスの取得, 1096
リモートサーバーへのログインの削除, 713
リモートサーバーへの割り当て, 548
ログインポリシー
ALTER LOGIN POLICY 文, 424
COMMENT 文を使用してコメントを追加, 512
CREATE LOGIN POLICY 文, 573
DROP LOGIN POLICY 文, 716
locked オプション, 573
max_connections オプション, 573
max_days_since_login オプション, 573
max_failed_login_attempts オプション, 573
max_non_dba_connections, 573
password_expiry_on_next_login オプション, 573
password_grace_time オプション, 573
password_life_time オプション, 573
デフォルトのログインポリシー, 573
ログインポリシーの作成
CREATE LOGIN POLICY 文, 573
ログファイル
ALTER DBSPACE 文を使用して領域を割り付け, 414
sa_get_request_profile での要求ログの分析, 1091
sa_get_request_times での要求ログの分析, 1092
使用可能な領域の決定, 1080
ロック
機能, 830
種類, 1112
テーブル, 831
表示, 1111
ブロック, 1055
ロックされたアカウント
原因の特定, 1096
論理演算子
3 値的論理, 59

構文, 9
ロー
SELECT 文, 895
UNLOAD 文を使用したデータのアンロード, 966
返される数の制限, 895
カーソルから削除, 695
カーソルからフェッチ, 749
カーソルを使用した挿入, 860
更新, 976
データベースから削除, 697
テーブルからすべてのローを削除, 961
テーブルへ挿入, 802
バルク挿入, 814
ローカルテンポラリテーブル
作成, 691
ローカルテンポラリファイル
CREATE LOCAL TEMPORARY TABLE 文を使用して作成, 571
ローカル変数
構文, 77
定義, 76
ローコンストラクターアルゴリズム
DUMMY システムテーブル, 993
ロージェネレーター
RowGenerator テーブル (dbo), 1027
ロージェネレーター
sa_rowgenerator システムプロシージャー, 1163
ロー数
システムビュー, 1311
ロー制限
説明, 895
ロード
LOAD TABLE 文, 814
st_geometry_load_shapefile システムプロシージャーを使用するシェイプファイル, 1235
クライアントコンピューター上のファイルからデータをロード, 816
コストモデル, 1109
指定された値からデータをロード, 817
データベースサーバーコンピューター上のファイルからデータをロード, 816
テーブルが即時ビューから参照されていると失敗, 826
テーブルに即時テキストインデックスがあると失敗, 826
バルク挿入, 814
ロールバック
トランザクション, 892
トランザクションのセーブポイント, 891
トリガー, 893
ローロック
sa_locks システムプロシージャー, 1112

わ

ワイド挿入
説明, 743
ワイルドカード
LIKE 探索条件, 42
PATINDEX 関数, 297
REGEXP 探索条件, 47
SIMILAR TO 探索条件, 48
式のワイルドカード, 18
割り当て
リモートサーバーへのログイン, 548
割り付け
ALTER DBSPACE 文を使用してディスク領域を割り付け, 414
記述子領域のメモリ, 406
割り付けの解除
記述子領域, 684