SYBASE iAnywhere

SQL Anywhere サーバ
SQL リファレンス

改訂 2007 年 3 月

版権と商標

目次

はじめに

このマニュアルの内容

このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。

特定の作業を行うための手順については、他のマニュアルでより具体的に説明されています。このマニュアルでは、使用可能な SQL 構文、システム・オブジェクトなどの全リストを参照できます。

対象読者

このマニュアルは、SQL Anywhere のすべてのユーザを対象としています。特に、Mobile Link、Ultra Light、SQL Remote のユーザにとって関心のある情報が掲載されています。他のマニュアルと一緒に使用するように構成されています。

SQL Anywhere のマニュアル

このマニュアルは、SQL Anywhere のマニュアル・セットの一部です。この項では、マニュアル・セットに含まれる各マニュアルと使用法について説明します。

SQL Anywhere のマニュアル

SQL Anywhere の完全なマニュアルは、各マニュアルをまとめたオンライン形式とマニュアル別の PDF ファイルで提供されます。いずれの形式のマニュアルも、同じ情報が含まれ、次のマニュアルから構成されます。

- **『SQL Anywhere 10 - 紹介』** このマニュアルでは、データの管理および交換機能を提供する包括的なパッケージである SQL Anywhere 10 について説明します。SQL Anywhere を使用すると、サーバ環境、デスクトップ環境、モバイル環境、リモート・オフィス環境に適したデータベース・ベースのアプリケーションを迅速に開発できるようになります。
- **『SQL Anywhere 10 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 10 とそれ以前のバージョンに含まれる新機能について説明します。
- **『SQL Anywhere サーバ - データベース管理』** このマニュアルでは、SQL Anywhere データベースの実行、管理、設定について説明します。管理ユーティリティとオプションのほか、データベース接続、データベース・サーバ、データベース・ファイル、バックアップ・プロシージャ、セキュリティ、高可用性、Replication Server を使用したレプリケーションについて説明します。
- **『SQL Anywhere サーバ - SQL の使用法』** このマニュアルでは、データベースの設計と作成の方法、データのインポート・エクスポート・変更の方法、データの検索方法、ストアド・プロシージャとトリガの構築方法について説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java プログラミング言語、Visual Studio .NET を使用してデータベース・アプリケーションを構築、配備する方法について説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。
- **『SQL Anywhere 10 - エラー・メッセージ』** このマニュアルでは、SQL Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。
- **『Mobile Link - クイック・スタート』** このマニュアルでは、セッションベースのリレーショナル・データベース同期システムである Mobile Link について説明します。Mobile Link テクノロジは、双方向レプリケーションを可能にし、モバイル・コンピューティング環境に非常に適しています。
- **『Mobile Link - サーバ管理』** このマニュアルでは、Mobile Link アプリケーションを設定して管理する方法について説明します。

- **『Mobile Link - クライアント管理』** このマニュアルでは、Mobile Link クライアントを設定、構成、同期する方法について説明します。Mobile Link クライアントには、SQL Anywhere または Ultra Light のいずれかのデータベースを使用できます。
- **『Mobile Link - サーバ起動同期』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。
- **『QAnywhere』** このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームでもあります。
- **『SQL Remote』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて説明します。このシステムによって、SQL Anywhere の統合データベースと複数の SQL Anywhere リモート・データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere 10 - コンテキスト別ヘルプ』** このマニュアルには、[接続] ダイアログ、クエリ・エディタ、Mobile Link モニタ、SQL Anywhere コンソール・ユーティリティ、インデックス・コンサルタント、Interactive SQL のコンテキスト別のヘルプが収録されています。
- **『Ultra Light - データベース管理とリファレンス』** このマニュアルでは、小型デバイス用 Ultra Light データベース・システムの概要を説明します。
- **『Ultra Light - AppForge プログラミング』** このマニュアルでは、Ultra Light for AppForge について説明します。Ultra Light for AppForge を使用すると、Palm OS、Symbian OS、または Windows CE を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - .NET プログラミング』** このマニュアルでは、Ultra Light.NET について説明します。Ultra Light.NET を使用すると、PC、ハンドヘルド、モバイル、埋め込みデバイスのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light - M-Business Anywhere プログラミング』** このマニュアルは、Ultra Light for M-Business Anywhere について説明します。Ultra Light for M-Business Anywhere を使用すると、Palm OS、Windows CE、または Windows XP を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - C/C++ プログラミング』** このマニュアルでは、Ultra Light C および Ultra Light C++ のプログラミング・インタフェースについて説明します。Ultra Light を使用すると、ハンドヘルド、モバイル、埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。

マニュアルの形式

SQL Anywhere のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含

まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

UNIX オペレーティング・システムでオンライン・マニュアルにアクセスするには、SQL Anywhere のインストール・ディレクトリまたはインストール CD に保存されている HTML マニュアルを参照してください。

- **PDF ファイル** SQL Anywhere の完全なマニュアル・セットは、Adobe Reader で表示できる Adobe Portable Document Format (pdf) 形式のファイルとして提供されています。

  Windows では、PDF 形式のマニュアルはオンライン・マニュアルの各ページ上部にある PDF のリンクから、または Windows の [スタート] メニュー ([スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル - PDF フォーマット]) からアクセスできます。

  UNIX では、PDF 形式のマニュアルはインストール CD にあります。

表記の規則

この項では、このマニュアルで使用されている書体およびグラフィック表現の規則について説明します。

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 ALTER TABLE のように大文字で表記します。

  ALTER TABLE [*owner.*]*table-name*

- **プレースホルダ**　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* のように表記します。

  ALTER TABLE [*owner.*]*table-name*

- **繰り返し項目**　繰り返し項目のリストは、次の例に示す *column-constraint* のように、リストの要素の後ろに省略記号 (ピリオド 3 つ …) を付けて表します。

  ADD *column-definition* [*column-constraint*, …]

  複数の要素を指定できます。複数の要素を指定する場合は、各要素間をカンマで区切る必要があります。

- **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。

  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。

- **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、項目間を縦線で区切り、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 つを必ず選択しなければならない場合は、選択肢を中カッコで囲み、縦棒で区切ります。

  [**QUOTES** { **ON** | **OFF** }]

  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

オペレーティング・システムの表記規則

- **Windows**　デスクトップおよびラップトップ・コンピュータ用の Microsoft Windows オペレーティング・システムのファミリのことです。Windows ファミリには Windows Vista や Windows XP も含まれます。
- **Windows CE**　Microsoft Windows CE モジュラ・オペレーティング・システムに基づいて構築されたプラットフォームです。Windows Mobile や Windows Embedded CE などのプラットフォームが含まれます。

  Windows Mobile は Windows CE 上に構築されています。これにより、Windows のユーザ・インタフェースや、Word や Excel といったアプリケーションの小規模バージョンなどの追加機能が実現されています。Windows Mobile は、モバイル・デバイスで最も広く使用されています。

  SQL Anywhere の制限事項や相違点は、基盤となっているオペレーティング・システム (Windows CE) に由来しており、使用しているプラットフォーム (Windows Mobile など) に依存していることはほとんどありません。
- **UNIX**　特に記述がないかぎり、UNIX は Linux プラットフォームと UNIX プラットフォームの両方のことです。

ファイルの命名規則

マニュアルでは、パス名やファイル名などのオペレーティング・システムに依存するタスクと機能を表すときは、通常 Windows の表記規則が使用されます。ほとんどの場合、他のオペレーティング・システムで使用される構文に簡単に変換できます。

- **ディレクトリ名とパス名**　マニュアルでは、ドライブを示すコロンや、ディレクトリの区切り文字として使用する円記号など、Windows の表記規則を使用して、ディレクトリ・パスのリストを示します。次に例を示します。

  ```
  MobiLink¥redirector
  ```

  UNIX、Linux、Mac OS X では、代わりにスラッシュを使用してください。次に例を示します。

  ```
  MobiLink/redirector
  ```

  SQL Anywhere がマルチプラットフォーム環境で使用されている場合、プラットフォーム間でのパス名の違いに注意する必要があります。
- **実行ファイル**　マニュアルでは、実行ファイルの名前は、Windows の表記規則が使用され、拡張子 *.exe* が付きます。UNIX、Linux、Mac OS X では、実行ファイルの名前には拡張子は付きません。NetWare では、実行ファイルの名前には、拡張子 *.nlm* が付きます。

  たとえば、Windows では、ネットワーク・データベース・サーバは *dbsrv10.exe* です。UNIX、Linux、Mac OS X では、*dbsrv10* になります。NetWare では、*dbsrv10.nlm* になります。
- **install-dir**　インストール・プロセスでは、SQL Anywhere をインストールするロケーションを選択できます。マニュアルでは、このロケーションは *install-dir* という表記で示されます。

インストールが完了すると、環境変数 SQLANY10 によって SQL Anywhere コンポーネントがあるインストール・ディレクトリのロケーション (*install-dir*) が指定されます。SQLANYSH10 は、SQL Anywhere が他の Sybase アプリケーションと共有しているコンポーネントがあるディレクトリのロケーションを指定します。

オペレーティング・システム別の *install-dir* のデフォルト・ロケーションの詳細については、「SQLANY10 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **samples-dir** インストール・プロセスでは、SQL Anywhere に含まれるサンプルをインストールするロケーションを選択できます。マニュアルでは、このロケーションは *samples-dir* という表記で示されます。

  インストールが完了すると、環境変数 SQLANYSAMP10 によってサンプルがあるディレクトリのロケーション (*samples-dir*) が指定されます。Windows の [スタート] メニューから、[プログラム] – [SQL Anywhere 10] – [サンプル・アプリケーションおよびプロジェクト] を選択すると、このディレクトリで [Windows エクスプローラ] ウィンドウが表示されます。

  オペレーティング・システム別の *samples-dir* のデフォルト・ロケーションの詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **環境変数** マニュアルでは、環境変数設定が引用されます。Windows では、環境変数を参照するのに、構文 *%envvar%* が使用されます。UNIX、Linux、Mac OS X では、環境変数を参照するのに、構文 *$envvar* または *${envvar}* が使用されます。

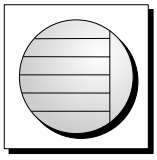
  UNIX、Linux、Mac OS X 環境変数は、*.cshrc* や *.tcshrc* などのシェルとログイン・スタートアップ・ファイルに格納されます。

グラフィック・アイコン

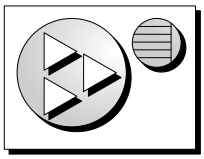
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

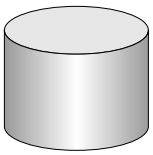
- SQL Anywhere などのデータベース・サーバ



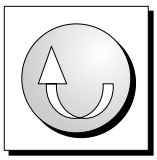
- Ultra Light アプリケーション



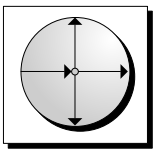
- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



- レプリケーションまたは同期のミドルウェア。ソフトウェアのこれらの部分は、データベース間のデータ共有を支援します。たとえば、Mobile Link サーバ、SQL Remote Message Agent などが挙げられます。



- Sybase Replication Server



- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

詳しい情報やリソース (コード交換など) については、iAnywhere Developer Network (http://www.ianywhere.com/developer/) を参照してください。

ご質問がある場合や支援が必要な場合は、次に示す Sybase iAnywhere ニュースグループのいずれかにメッセージをお寄せください。

ニュースグループにメッセージをお送りいただく際には、ご使用の SQL Anywhere バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng10 -v** と入力して確認できます。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります (ニュースグループにおけるサービスは英語でのみの提供となります)。以下のニュースグループがあります。

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

ニュースグループに関するお断り
iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。
iAnywhere のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

マニュアルに関するご意見、ご提案は、SQL Anywhere ドキュメンテーション・チームの iasdoc@ianywhere.com 宛てに電子メールでお寄せください。このアドレスに送信された電子メールに返信はいたしませんが、お寄せいただいたご意見、ご提案は必ず読ませていただきます。

マニュアルまたはソフトウェアについてのフィードバックは、上記のニュースグループを通してお寄せいただいてもかまいません。

パート I. SQL の使用

この項では、データ型、関数、文を含む、SQL Anywhere の SQL 言語について説明します。

第 1 章

SQL 言語の要素

目次

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、二重引用符で囲みます。SQL 文で使用するキーワードのうち、すべてではありませんがその多くが予約語です。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 はキーワードの大文字と小文字を区別しないため、次の各語は大文字、小文字、またはその組み合わせで使用できます。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。

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。

詳細については、「sql_needs_quotes 関数」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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	**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**	**index_lparen**
inner	**inout**	**insensitive**	**insert**
inserting	**install**	**instead**	**int**
integer	**integrated**	**intersect**	**into**
iq	**is**	**isolation**	**join**
kerberos	**key**	**lateral**	**left**
like	**lock**	**login**	**long**
match	**membership**	**message**	**mode**
modify	**natural**	**nchar**	**new**
no	**noholdlock**	**not**	**notify**

null	**numeric**	**nvarchar**	**of**
off	**on**	**open**	**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**	**sqlcode**	**sqlstate**
start	**stop**	**subtrans**	**subtransaction**
synchronize	**syntax_error**	**table**	**temporary**
then	**time**	**timestamp**	**tinyint**
to	**top**	**tran**	**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**	**with_cube**
with_lparen	**with_rollup**	**within**	**work**
writetext	**xml**		

識別子

識別子は、ユーザ 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た場合は、二重引用符または角カッコで囲みます。

- 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)
- 識別子に予約語が含まれる
- 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 (_)、アット記号 (@)、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。

- " (二重引用符)
- 制御文字 (0x20 未満の文字)
- ¥¥ (二重円記号)

1 つの円記号は、エスケープ文字として使用される場合のみ識別子で使用できます。

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。ただし角カッコは、quoted_identifier の設定にかかわらず、どのような場合でも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。

参照

- 予約語の完全なリストについては、「予約語」 4 ページを参照してください。
- quoted_identifier オプションの詳細については、「quoted_identifier オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

例

次の識別子はすべて有効です。

```
Surname
"Surname"
[Surname]
SomeBigName
"Client Number"
```

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。「バイナリ・リテラル」 9 ページを参照してください。
- 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 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。

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ー9 の数字、大文字と小文字の A ー 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ときは、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。

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。

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。

引用符で囲んだ値に N を前に付けることで、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ときに使用されます。そのため、文字列の一重引用符は、次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。

  ```
  'John''s database'
  ```

- 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。コード・ページ 1252 の次の例では、数字 1、2、3 に続けてユーロ通貨記号を表現しています。

  ```
  '123¥x80'
  ```


- 文字列内に改行を表すには、次のように円記号と n (¥n) を使用します。

  'First line:¥nSecond line:'

- バックスラッシュはエスケープ・シーケンスの開始を示すときに使用するため、文字列内のバックスラッシュ記号は、次のようにバックスラッシュを追加してエスケープする必要があります。

  'c:¥¥temp'

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ユニコード文字を使用する場合、UNISTR 関数を使用します。「UNISTR 関数 [文字列]」 274 ページを参照してください。

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。比較演算子の詳細については、「探索条件」 20 ページを参照してください。

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。

詳細については、「演算子の優先度」 14 ページを参照してください。

比較演算子

比較条件の構文は、次のとおりです。

expression compare expression

ここで、*compare* は比較演算子です。次の比較演算子を使用できます。

演算子	説明
=	等価
>	より大きい
<	より小さい
>=	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
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*

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。

文字列の比較時の大文字と小文字の区別の詳細については、「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ - データベース管理』を参照してください。

後続ブランク　文字列を比較するとき、SQL Anywhere の動作は -b オプションによって制御されます。このオプションは、データベースの作成時に設定されます。

ブランクの埋め込みの詳細については、「初期化ユーティリティ (dbinit)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

論理演算子

探索条件は AND、OR、NOT で結合できます。

AND を使用して、次のように条件を結合します。

condition1 **AND** *condition2*

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 を使用して、次のように条件を結合します。

condition1 **OR** *condition2*

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 演算子の構文は、次のとおりです。

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 演算子では、論理値をテストできます。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) と評価されれば条件は TRUE になります。それ以外の場合、値は FALSE です。

詳細については、「3 値的論理」 27 ページを参照してください。

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

- **モジュロ**　SQL Anywhere では、percent_as_comment オプションが Off に設定されている場合のみ、% 演算子を使用できます。デフォルト値は On です。

文字列演算子

expression || expression　文字列連結 (2 本の縦線)。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/2003**　|| 演算子は、SQL/2003 の文字列連結演算子です。

ビット処理演算子

SQL Anywhere では、次の演算子を整数データ型とビット配列データ型に使用できます。

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。

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

FROM 句にテーブル式を使用する SQL/2003 のジョイン構文がサポートされます。「FROM 句」 552 ページを参照してください。

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。「tsql_outer_joins オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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。

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*

パラメータ

case-expression:
CASE *expression*
WHEN *expression*
THEN *expression*,⋯
[**ELSE** *expression*]
END

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

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

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

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

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。「SQL 関数」 91 ページと「変数」 36 ページを参照してください。

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」 16 ページ
- 「特別値」 30 ページ
- 「式内のカラム名」 16 ページ
- 「SQL 関数」 91 ページ
- 「式内のサブクエリ」 17 ページ
- 「探索条件」 20 ページ
- 「SQL データ型」 47 ページ
- 「変数」 36 ページ
- 「CASE 式」 17 ページ

標準と互換性

- その他の相違点については、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。相関名の詳細については、「FROM 句」 552 ページを参照してください)。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"
```

識別子の詳細については、「識別子」 7 ページを参照してください。

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。

SELECT 文は、カッコで囲み、1 つの選択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。

サブクエリのその他の使用法については、「探索条件内のサブクエリ」 21 ページを参照してください。

IF 式

IF 式の構文は、次のとおりです。

IF *condition*
THEN *expression1*
[**ELSE** *expression2*]
ENDIF

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。
- 条件が UNKNOWN の場合、IF 式は NULL を返します。

TRUE、FALSE、UNKNOWN の各条件の詳細については、「NULL 値」 43 ページと「探索条件」 20 ページを参照してください。

IF 文は IF 式とは異なります。
IF 式の構文と IF 文の構文を混同しないでください。IF 文の詳細については、「IF 文」 580 ページを参照してください。

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*
WHEN *expression*

THEN *expression*, …
[**ELSE** *expression*]
END

CASE 文に続く式が WHEN 文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。

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*, …
[**ELSE** *expression*]
END

WHEN 文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'
    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 文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。

CASE 文は CASE 式とは異なります。
CASE 文の構文と CASE 式の構文を混同しないでください。CASE 文の詳細については、「CASE 文」 364 ページを参照してください。

式の互換性

区切り文字列のデフォルト解釈

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/2003 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 [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

quoted_identifier が off の場合、SQL の予約語は識別子として使用できません。

予約語の完全なリストについては、「予約語」 4 ページを参照してください。

オプションの設定

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

```
SET quoted_identifier On
```

SQL Anywhere の次の文では、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/2003 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 (SQL Anywhere のデフォルト設定) 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)
) ;
INSERT "user" ( col1 )
VALUES ( 'abcde' ) ;
```

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。

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


探索条件

探索条件は、WHERE 句、HAVING 句、CHECK 句、ジョインの ON 句、または IF 式に指定された基準です。

構文

search-condition:
expression compare expression
| *expression compare* { [**ANY** | **SOME**] | **ALL** } (*subquery*)
| *expression* **IS** [**NOT**] **NULL**
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **LIKE** *expression* [**ESCAPE** *expression*]
| *expression* [**NOT**] **IN** **(** { *expression*
| *subquery*
| *value-expr1* **,** ⋯ } **)**
| **EXISTS (** *subquery* **)**
| **NOT** *condition*
| *search-condition* **AND** *search-condition*
| *search-condition* **OR** *search-condition*
| **(** *search-condition* **)**
| **(** *search-condition* **,** *estimate* **)**
| *search-condition* **IS** [**NOT**] { **TRUE** | **FALSE** | **UNKNOWN** }
| *trigger-operation*

パラメータ

compare:
= | **>** | **<** | **>=** | **<=** | **<>** | **!=** | **!<** | **!>**

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

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。3 値的論理で論理演算子がどのように結合されるかを示した表については、「3 値的論理」 27 ページを参照してください。

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 UKNKNOWN または FALSE のローは、探索条件を満たしません。NULL の詳細については、「NULL 値」 43 ページを参照してください。

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。探索条件におけるサブクエリの詳細については、「探索条件内のサブクエリ」 21 ページを参照してください。

以降の項で、異なるタイプの探索条件について説明します。

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」 15 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。

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 (「比較演算子」 11 ページを参照) 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。

ALL、ANY、SOME 探索条件

ANY 探索条件

ANY 探索条件の構文は次のとおりです。

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

comparison-operator は <=、=、<、>、>=、<>、!<、!>、!= のいずれかです。

ANY の代わりにキーワード SOME を使用できます。

ANY 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 UKNOWN 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。

たとえば、等号演算子のある ANY 探索条件は次のように評価されます。

expression = **ANY (** *subquery* **)**

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 です。

注意
=ANY 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。

ALL 探索条件

ALL 探索条件の構文は次のとおりです。

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

comparison-operator は <=、=、<、>、>=、<>、!<、!>、!= のいずれかです。

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 UKNOWN 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。

BETWEEN 探索条件

BETWEEN 探索条件の構文は次のとおりです。

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

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、*expr* が *start-expr* と *end-expr* 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**] (*expr* >= *start-expr* **AND** *expr* <= *end-expr*)

LIKE 探索条件

LIKE 探索条件の構文は次のとおりです。

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

LIKE 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 *pattern* に一致すれば、探索条件は TRUE と評価されます。*expression* または *pattern* が NULL 値の場合、この探索条件は UNKNOWN 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。

pattern には、任意の数のワイルドカードを指定できます。次のワイルドカードを使用できます。

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 (例：a_)
% (パーセント記号)	0 個以上の文字からなる任意の文字列 (例：bl%)
[]	指定範囲内、または一連の指定文字の任意の 1 文字 (例：T[oi]m)
[^]	指定範囲外、または一連の指定文字*以外*の任意の 1 文字 (例：'M[^c]%')

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があるとします。

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

文字 a で始まり、末尾から 2 つ目の文字が b のローの場合、TRUE になります。

escape-expression が指定されている場合は、1 文字として評価されます。その文字は、*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。

長さ 126 文字以下のパターンは、すべてサポートされています。254 文字よりも長いパターンは、サポートされていません。127 ー 254 文字の長さのパターンは、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。

一連の文字の検索

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。たとえば、次の探索条件は文字列 *smith* と *smyth* を検出します。

```
LIKE 'sm[iy]th'
```

範囲内の文字の検索

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。たとえば、次の探索条件は、文字列 bough と rough を検出しますが、tough は検出しません。

```
LIKE '[a-r]ough'
```

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。照合内の文字の順序については、「国際言語と文字セット」『SQL Anywhere サーバ - データベース管理』を参照してください。

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、式 [z-a] がある LIKE 探索条件は、ローを返しません。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。

データベースが大文字と小文字を区別するように作成されていないかぎり、文字の範囲では大文字と小文字を区別しません。たとえば、次の探索条件は、文字列 Bough、rough、TOUGH を検出します。

```
LIKE '[a-z]ough'
```

データベースが大文字と小文字を区別するように作成されている場合は、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データベース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、次の探索条件は、文字列 Bough、rough、TOUGH を検出します。

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

文字範囲と一連の文字検索の結合

角カッコ内で、文字範囲と一連の文字を結合できます。たとえば、次の探索条件は、文字列 bough、rough、tough を検出します。

```
... LIKE '[a-rt]ough'
```

角かっこ [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 文字の検索

脱字記号 (^) は、検索から除外する文字範囲を指定します。たとえば、次の探索条件は文字列 tough を検出しますが、文字列 rough や bough は検出しません。

```
... LIKE '[^a-r]ough'
```

脱字記号は、この記号以外のカッコ内の内容をすべて否定します。たとえば、角かっこ [^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。

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。

- 式 [a-] は、文字 a または - に一致します。
- 式 [] 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。

- 式 [または [abp-q は、正しい式ではなく、構文エラーになります。
- 角カッコ内ではワイルドカードは使用できません。式 [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。式 [a^b] は a、^、または b のいずれかを検出します。

後続ブランクのある探索パターン

探索パターンに後続ブランクが含まれる場合、SQL Anywhere 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_' の探索パターンは、値 '90 ' には一致しますが、値 '90' には一致しません。

ブランクが埋め込まれたデータベース

LIKE 述部の LIKE パターンは、パターン一致表現です。ブランクが埋め込まれたデータベースかどうかによってこのセマンティックが変わることはありません。式を LIKE パターンと一致させるには、値の左から右に 1 文字ずつを LIKE パターンに対応させます。評価時に、値または式に追加のブランクは埋め込まれません。そのため、式 'a' は LIKE パターンの 'a' と一致しますが、LIKE パターンの 'a ' (a の後にスペースがある場合) や 'a_' とは一致しません。

標準と互換性

- ESCAPE 句がサポートされているのは、SQL Anywhere のみです。

IN 探索条件

IN 探索条件の構文は次のとおりです。

expression [**NOT**] **IN** { **(** *subquery* **)** | **(** *expression2* **)** | **(** *value-expr*, ⋯ **)** }

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、*subquery* 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 (** *values* **)** は *expression* = **ANY (** *values* **)** と同義です。

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

value-expr 引数は、単一値をとる式です。これには、文字列、数字、日付、または他の任意の SQL データ型などがあります。

EXISTS 探索条件

EXISTS 探索条件の構文は次のとおりです。

EXISTS(*subquery* **)**

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。

IS NULL および IS NOT NULL 探索条件

IS NULL 探索条件の構文は次のとおりです。

expression **IS** [**NOT**] **NULL**

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。

真理値探索条件

真理値探索条件の構文は次のとおりです。

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

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

- ベンダ拡張。

トリガ・オペレーション条件

トリガ・オペレーション条件の構文は、次のとおりです。

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

トリガ・オペレーション条件は、トリガのみで使用でき、トリガ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

次のトリガは、トリガを起動させたアクションを示すメッセージを表示します。

```
CREATE TRIGGER tr BEFORE INSERT, UPDATE, DELETE
ON sample_table
REFERENCING OLD AS t1old
FOR EACH ROW
BEGIN
   DECLARE msg varchar(255);

   SET msg = 'This trigger was fired by an ';
   IF INSERTING THEN
      SET msg = msg || 'insert'
   ELSEIF DELETING THEN
      set msg = msg || 'delete'
   ELSEIF UPDATING THEN
      set msg = msg || 'update'
   END IF;
   MESSAGE msg TO CLIENT
END
```

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

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 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。詳細については、「FROM 句」 552 ページを参照してください。

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。

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。

ユーザ定義の選択性推定を無効にする方法の詳細については、「user_estimates オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

統計の詳細については、「オプティマイザの推定とカラム統計」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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)
```

小数を使用すれば、特に大きなテーブルの場合、ジョインのユーザ予測値はさらに正確になります。

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、**user**、**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

STRING

参照

- 「式」 15 ページ

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

- 「式」 15 ページ
- 「TIME データ型」 72 ページ

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・ユーザ ID を含む文字列を返します。

データ型

STRING

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

- 「式」 15 ページ
- 「SQL Remote インストール環境の設計」 『SQL Remote』

CURRENT TIME 特別値

現在の時、分、秒 (小数位あり) で構成される時刻を返します。

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」 15 ページ
- 「TIME データ型」 72 ページ

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。秒は小数第 3 位まで格納されます。精度はシステム・クロックの精度によって制限されます。

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。

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。

データ型

TIMESTAMP

参照

- 「CURRENT TIME 特別値」 31 ページ
- 「TIMESTAMP 特別値」 34 ページ
- 「式」 15 ページ
- 「TIMESTAMP データ型」 73 ページ
- 「GETDATE 関数 [日付と時刻]」 170 ページ
- 「NOW 関数 [日付と時刻]」 211 ページ

CURRENT USER 特別値

CURRENT USER は、現在の接続のユーザ ID を含む文字列を返します。

データ型

STRING

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、CURRENT USER のデフォルト値の入ったカラムは変更されません。
CURRENT_USER は、CURRENT USER と同じです。

参照

- 「式」 15 ページ

CURRENT UTC TIMESTAMP 特別値

CURRENT UTC TIMESTAMP は、CURRENT DATE と CURRENT TIME を組み合わせ、サーバのタイムゾーン調整値で調整すると、年、月、日、時、分、秒、秒の小数位で構成される協定世界時 (UTC: Coordinated Universal Time) TIMESTAMP 値が得られます。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。

データ型

TIMESTAMP

参照

- 「TIMESTAMP データ型」 73 ページ
- 「UTC TIMESTAMP 特別値」 35 ページ
- 「CURRENT TIMESTAMP 特別値」 31 ページ
- 「truncate_timestamp_values オプション [データベース] [Mobile Link クライアント]」 『SQL Anywhere サーバ - データベース管理』

LAST USER 特別値

LAST USER は、ローを最後に更新したユーザの名前です。

データ型

STRING

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と日時の両方を記録できます (ただし、別々のローに記録されます)。

参照

- 「CURRENT USER 特別値」 32 ページ
- 「CURRENT TIMESTAMP 特別値」 31 ページ
- 「CREATE TABLE 文」 460 ページ

SQLCODE 特別値

SQLCODE は、現在の SQLCODE 値です。

データ型

文字列

備考

SQLCODE 値は各文の後に設定されます。SQLCODE をチェックして、文の実行が成功したかどうかを確認することができます。

参照

- 「式」 15 ページ
- SQL Anywhere 10 - エラー・メッセージ 『SQL Anywhere 10 - エラー・メッセージ』.

SQLSTATE 特別値

SQLSTATE は、現在の SQLSTATE 値です。

データ型

STRING

備考

SQLSTATE 値は各文の後に設定されます。SQLSTATE をチェックして、文の実行が成功したかどうかを確認することができます。

参照

- 「式」 15 ページ
- SQL Anywhere 10 - エラー・メッセージ 『SQL Anywhere 10 - エラー・メッセージ』

TIMESTAMP 特別値

TIMESTAMP は、テーブルの各ローが最後に修正された時刻を示します。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。

データ型

TIMESTAMP

備考

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

- 「TIMESTAMP データ型」 73 ページ
- 「CURRENT TIMESTAMP 特別値」 31 ページ
- 「CURRENT UTC TIMESTAMP 特別値」 32 ページ
- 「default_timestamp_increment オプション [データベース] [Mobile Link クライアント]」 『SQL Anywhere サーバ - データベース管理』
- 「truncate_timestamp_values オプション [データベース] [Mobile Link クライアント]」 『SQL Anywhere サーバ - データベース管理』

USER 特別値

USER は、現在の接続のユーザ ID を含む文字列を返します。

データ型

STRING

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。

備考

UPDATE 時には、USER のデフォルト値の入ったカラムは変更されません。

参照

- 「式」 15 ページ
- 「CURRENT USER 特別値」 32 ページ
- 「LAST USER 特別値」 32 ページ

UTC TIMESTAMP 特別値

UTC TIMESTAMP は、テーブルの各ローが最後に修正されたときの協定世界時 (UTC) を示します。

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。

データ型

TIMESTAMP

備考

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 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 オプションを使用して、SQL Anywhere の UTC 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。

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

参照

- 「TIMESTAMP データ型」 73 ページ
- 「CURRENT UTC TIMESTAMP 特別値」 32 ページ
- 「TIMESTAMP 特別値」 34 ページ
- 「default_timestamp_increment オプション [データベース] [Mobile Link クライアント]」 『SQL Anywhere サーバ - データベース管理』
- 「truncate_timestamp_values オプション [データベース] [Mobile Link クライアント]」 『SQL Anywhere サーバ - データベース管理』

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・サーバの現在のバージョン番号です。ユーザ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、ユーザが宣言します。これらは、プロシージャや 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」 73 ページ
- 「CREATE VARIABLE 文」 480 ページ

ローカル変数

SQL Anywhere ではローカル変数をサポートしています。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、プロシージャが複合文中から呼び出されるかぎり、プロシージャ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、[Interactive SQL メッセージ] タブにメッセージ local_var = 10 が表示されます。

- 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、「**カラムが見つかりません**」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、サーバ・メッセージ・ウィンドウにメッセージ local_var = 10 が表示されます。

バッチとローカル変数のスコープの詳細については、「Transact-SQL プロシージャの中の変数」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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、プロシージャにパラメータ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、サーバ・ウィンドウにメッセージ con_var = 10 が表示されます。

グローバル変数

グローバル変数は、データベース・サーバ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殊定数

特殊定数 (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;
```

プロシージャとトリガでは、グローバル変数は変数リストから選択できます。次のプロシージャでは、*ver* パラメータ内にあるサーバ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。

グローバル変数のリスト

次の表は、SQL Anywhere で使用できるグローバル変数のリストです。Transact-SQL と互換性を保つために用意されているグローバル変数もあります。このようなグローバル変数は、0、1、または NULL のうちいずれかの固定値を返します。

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)
@@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

変数名	意味
@@error	通常、直前に実行された文のエラー・ステータス (成功または失敗) のチェックに使用する。前のトランザクションが成功した場合、値は 0。それ以外の場合は、システムが生成した最新のエラー番号。エラーが発生した場合、if @@error != 0 return のような文によって終了する。PRINT 文や IF テストなどを含む、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・チェックは成功したかどうかを調査する文の直後に行う。
@@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する。この機能は、異なった値を返す以外は、@@sqlstatus と同じ。Microsoft SQL Server との互換性のために用意されている。@@fetch_status は、次のいずれかの値を格納する。 ◆ 0 フェッチ文は正常終了した。 ◆ -1 フェッチ文エラーになった。 ◆ -2 結果セットには、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新の値 「@@identity グローバル変数」 41 ページを参照してください。
@@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)
@@isolation	接続の現在の独立性レベル。@@isolation は、アクティブ・レベルの値をとる。
@@langid	現在の接続で使用中の言語のユニークな言語 ID。
@@language	接続で使用中の言語の名前を返す。
@@max_connections	パーソナル・サーバの場合、サーバへの同時接続最大数。値は 10。ネットワーク・サーバの場合、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る)。
@@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	現在実行中のプロシージャのストアド・プロシージャ 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、文の直後に確認する。 INSERT、UPDATE、DELETE は、@@rowcount を影響されたローの数に設定する。 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ない。
@@servername	現在のデータベース・サーバの名前
@@spid	現在の接続の接続ハンドル。これは、sa_conn_info プロシージャによって表示されるものと同じ値。
@@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する。@@sqlstatus は、次のいずれかの値を格納する。 ◆ 0 フェッチ文は正常終了した。 ◆ 1 フェッチ文エラーになった。 ◆ 2 結果セットには、これ以上データがない。
@@textsize	SET TEXTSIZE オプションの現在の値。select 文によって返される text または image データの最大長 (バイト単位)。デフォルト設定は 32765。これは、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 文によって設定できる。
@@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)
@@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 0、連鎖の場合 1。
@@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・カウントが増加する。
@@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

@@identity グローバル変数

@@identity 変数には、IDENTITY カラムまたは DEFAULT 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。ローがテーブルに挿入されると、@@identity の値は必ずリセットされます。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

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には、T1 に挿入された値が返されます。トリガ内では、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。トリガが両方の値にアクセスする場合、トリガはそれらの値をローカル変数にコピーできます。

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・サーバは、コメントを実行しません。

SQL Anywhere では、次の数種類のコメント・インジケータを使用できます。

- **-- (二重ハイフン)** データベース・サーバは、この行の残りの文字を無視します。これは、SQL/2003 のコメント・インジケータです。
- **// (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。
- **/* … */ (スラッシュ – アスタリスク)** 2 つのコメント・マーカの間にある文字は、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。
- **% (パーセント記号)** パーセント記号は、percent_as_comment オプションが On に設定されていれば、二重ハイフンと同じ意味になります。% を、コメント・インジケータとして使用しないことをおすすめします。

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
```

NULL 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な値です。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	フィールドには値があるが、不定である
不適切	フィールドは、この特定のローには適さない

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。

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)。この句の詳細については、「探索条件」 20 ページを参照してください。

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は、すべて結果は 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 (「SQL 関数」 91 ページを参照してください)。

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。探索条件の結合における 3 値的論理の影響の詳細については、「探索条件」 20 ページを参照してください。

セット演算子と DISTINCT 句

セット演算 (UNION、INTERSECT、EXCEPT) と DISTINCT 演算では、探索条件の場合と NULL の扱いが異なります。ローに NULL が含まれる場合、それ以外の場合でローが同じ場合、ローはこれらの演算の目的に対して同様に扱われます。

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/2003**　コア機能。
- **Sybase**　Adaptive Server Enterprise では、一部のコンテキストで NULL が値として扱われますが、SQL Anywhere では値として扱われません。たとえば、SQL Anywhere の場合、次の WHERE 句では、NULL であるカラム c1 のローはクエリの結果に含まれません。これは、条件に値 UNKNOWN があるためです。

  ```
  WHERE NOT( C1 = NULL )
  ```

  Adaptive Server Enterprise の場合は、条件が TRUE と評価され、カラムのローが返されます。互換性を保つには、比較演算子ではなく IS NULL を使用してください。

  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。

詳細については、「tds_empty_string_is_null オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

参照

- 「式」 15 ページ
- 「探索条件」 20 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' )
```

第 2 章

SQL データ型

目次

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**　ユニコードの UTF-8 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ユニコード・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」 460 ページ
- 「string_rtruncation オプション [互換性]」 『SQL Anywhere サーバ - 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

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 ー 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。文字長のセマンティックを使用する場合、データベースのエンコーディングで、文字長に文字の最大長を乗じた数が 32767 バイトを超えないようにします。次の表は、サポートされる文字セットの型ごとの最大長です。

文字セット	CHAR の最大長
SBCS	32767 バイト
DBCS	16383 バイト
UTF-8	8191 バイト

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。

文字長のセマンティックを使用すると、使用するインタフェースによって、クライアント・アプリケーションでカラムに DESCRIBE が実行されたときに返される値が変わることがあります。たとえば、Embedded SQL クライアントで、バイト長のセマンティックを使用して宣言されたカラムに DESCRIBE が実行された場合、バイト長が指定された値が返されます。結果として、CHAR(10) のカラムは 10 バイト長の DT_FIXCHAR 型として記述されます。ただし、Embedded SQL クライアントで、文字長のセマンティックを使用して宣言されたカラムに DESCRIBE が実行された場合、クライアントの CHAR 文字セットの最大バイト長が返されます。たとえば、CHAR 文字セットに UTF-8 を使用する Embedded SQL クライアントの場合、CHAR(10 CHAR) カラムは 40 バイト長 (10 文字の各文字に最高 4 バイトを乗じた値) の DT_FIXCHAR 型と記述されます。

参照

- 「VARCHAR データ型」 54 ページ
- 「LONG VARCHAR データ型」 50 ページ
- 「NCHAR データ型」 50 ページ

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。文字長のセマンティックはベンダ拡張です。

LONG NVARCHAR データ型

LONG NVARCHAR データ型は、任意の長さのユニコード文字データを格納します。

構文

LONG NVARCHAR

備考

最大値は 2 GB です。

文字は UTF-8 に格納されます。各文字には 1 ー 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。「db_change_nchar_charset 関数」『SQL Anywhere サーバ - プログラミング』を参照してください。

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

参照

- 「NCHAR データ型」 50 ページ
- 「NVARCHAR データ型」 52 ページ
- 「LONG VARCHAR データ型」 50 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

最大値は 2 GB です。

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」 48 ページ
- 「VARCHAR データ型」 54 ページ
- 「LONG NVARCHAR データ型」 49 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

NCHAR データ型

NCHAR データ型は、8191 文字までのユニコード文字データを格納します。

構文

NCHAR [**(** *max-length* **)**]

パラメータ

max-length　文字列のバイト単位での最大長。長さの範囲は 1 ー 8191 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。

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。「db_change_nchar_charset 関数」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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。

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 では、odbc_distinguish_char_and_varchar オプションに応じて、NCHAR は SQL_WCHAR または SQL_WVARCHAR として記述されます。「odbc_distinguish_char_and_varchar オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

参照

- 「CHAR データ型」 48 ページ
- 「NVARCHAR データ型」 52 ページ
- 「LONG NVARCHAR データ型」 49 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

NTEXT データ型

NTEXT データ型は、任意の長さのユニコード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」 49 ページ
- 「TEXT データ型」 53 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

NVARCHAR データ型

NVARCHAR データ型は、8191 文字までのユニコード文字データを格納します。

構文

NVARCHAR [**(** *max-length* **)**]

パラメータ

max-length 文字列のバイト単位での最大長。長さの範囲は 1 ー 8191 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。「db_change_nchar_charset 関数」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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 NVARCHAR 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。

ODBC では、odbc_distinguish_char_and_varchar オプションに応じて、NVARCHAR は SQL_WCHAR または SQL_WVARCHAR として記述されます。「odbc_distinguish_char_and_varchar オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

参照

- 「NCHAR データ型」 50 ページ
- 「LONG NVARCHAR データ型」 49 ページ
- 「VARCHAR データ型」 54 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

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」 50 ページ
- 「NTEXT データ型」 51 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

UNIQUEIDENTIFIERSTR データ型

UNIQUEIDENTIFIERSTR データ型は、CHAR(36) として実装されるドメインです。

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 - SQL の使用法』
- 「STRTOUUID 関数 [文字列]」 263 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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 ー 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。文字長のセマンティックを使用する場合、データベースのエンコーディングで、文字長に文字の最大長を乗じた数が 32767 バイトを超えないようにします。次の表は、サポートされる文字セットの型ごとの最大長です。

文字セット	VARCHAR の最大長
SBCS	32767 バイト
DBCS	16383 バイト
UTF-8	8191 バイト

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。

文字長のセマンティックを使用すると、使用するインタフェースによって、クライアント・アプリケーションでカラムに DESCRIBE が実行されたときに返される値が変わることがあります。たとえば、Embedded SQL クライアント・アプリケーションで、バイト長のセマンティックを使用して宣言されたカラムに DESCRIBE が実行された場合、バイト長が指定された値が返されます。結果として、VARCHAR(10) のカラムは 10 バイト長の DT_VARCHAR 型として記述されます。ただし、Embedded SQL クライアント・アプリケーションで、文字長のセマンティックを使用して宣言されたカラムに DESCRIBE が実行された場合、クライアントの CHAR 文字セットの最大バイト長が返されます。たとえば、CHAR 文字セットに UTF-8 を使用するクライアントの場合、VARCHAR(10 CHAR) カラムは 40 バイト長 (10 文字の各文字に最高 4 バイトを乗じた値) の DT_VARCHAR 型と記述されます。

参照

- 「CHAR データ型」 48 ページ
- 「LONG VARCHAR データ型」 50 ページ
- 「NVARCHAR データ型」 52 ページ

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。文字長のセマンティックはベンダ拡張です。

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

最大値は 2 GB です。

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、適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。

XML 要素を生成するときの XML データ型の使用の詳細については、「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 - SQL の使用法』を参照してください。

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 - SQL の使用法』

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。

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 [データベース]」『SQL Anywhere サーバ - データベース管理』と「scale オプション [データベース]」『SQL Anywhere サーバ - 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 ー 2^{64} − 1、または 0 ー 18446744073709551615 です。

デフォルトでは、このデータ型は符号付きです。

参照

- 「BIT データ型」 57 ページ
- 「INTEGER データ型」 60 ページ
- 「SMALLINT データ型」 62 ページ
- 「TINYINT データ型」 63 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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。

参照

- 「BIGINT データ型」 56 ページ
- 「INTEGER データ型」 60 ページ
- 「SMALLINT データ型」 62 ページ
- 「TINYINT データ型」 63 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

precision　式の桁数を指定する整数式。1 ー 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 ー 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。詳細については、「precision オプション [データベース]」『SQL Anywhere サーバ - データベース管理』と「scale オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

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。

セマンティック上、DECIMAL は NUMERIC と同じです。

参照

- 「FLOAT データ型」 59 ページ
- 「REAL データ型」 61 ページ
- 「DOUBLE データ型」 58 ページ
- 「NUMERIC データ型」 61 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003**　SQL/2003 と互換性があります。

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 2.22507385850721e−308 ー 1.79769313486231e+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、15 桁を超えると丸め誤差が出ます。

参照

- 「FLOAT データ型」 59 ページ
- 「REAL データ型」 61 ページ
- 「DECIMAL データ型」 57 ページ
- 「NUMERIC データ型」 61 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ
- 「数値セット間の変換」 86 ページ

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。

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

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。

「float_as_double オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を使用して、Adaptive Server Enterprise との互換性に対する FLOAT データ型の動作を調節できます。

参照

- 「DOUBLE データ型」 58 ページ
- 「REAL データ型」 61 ページ
- 「DECIMAL データ型」 57 ページ
- 「NUMERIC データ型」 61 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。

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。

参照

- 「BIGINT データ型」 56 ページ
- 「BIT データ型」 57 ページ
- 「SMALLINT データ型」 62 ページ
- 「TINYINT データ型」 63 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。UNSIGNED キーワードはベンダ拡張です。

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

precision　式の桁数を指定する整数式。1 ー 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 ー 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。詳細については、「precision オプション [データベース]」『SQL Anywhere サーバ - データベース管理』と「scale オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

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 )
```

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。

セマンティック上、NUMERIC は DECIMAL と同じです。

参照

- 「FLOAT データ型」 59 ページ
- 「REAL データ型」 61 ページ
- 「DOUBLE データ型」 58 ページ
- 「DECIMAL データ型」 57 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ
- 「数値セット間の変換」 86 ページ

標準と互換性

- **SQL/2003**　scale オプションが 0 に設定されている場合、SQL/2003 と互換性があります。

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 データ型は概数値データ型です。算術演算後に丸め誤差がでます。

値の範囲は -3.402823e+38 ー 3.402823e+38 であり、ゼロに最も近い最小の数値は 1.175495e-38 です。REAL として保持される値は、厳密には 10 有効桁数ですが、6 桁を越えると丸め誤差が出ます。

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。

参照

- 「DOUBLE データ型」 58 ページ
- 「FLOAT データ型」 59 ページ
- 「DECIMAL データ型」 57 ページ
- 「NUMERIC データ型」 61 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。

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} ー $2^{15} - 1$、または −32768 ー 32767 です。

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

参照

- 「BIGINT データ型」 56 ページ
- 「BIT データ型」 57 ページ
- 「INTEGER データ型」 60 ページ
- 「TINYINT データ型」 63 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** SQL/2003 と互換性があります。UNSIGNED キーワードはベンダ拡張です。

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 ー 2^8 −1、または 0 ー 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。

参照

- 「BIGINT データ型」 56 ページ
- 「BIT データ型」 57 ページ
- 「INTEGER データ型」 60 ページ
- 「SMALLINT データ型」 62 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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) のドメインとして実装されます。

参照

- 「SMALLMONEY データ型」 64 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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) のドメインとして実装されます。

参照

- 「MONEY データ型」 64 ページ
- 「数値関数」 98 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

ビット配列データ型

ビット配列は、bit データ (0 と 1) を格納するために使用します。

ビット配列データ型はビットの配列を格納するときに使用します。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

- 「VARBIT データ型」 65 ページ
- 「ビット配列の変換」 85 ページ
- 「ビット配列関数」 94 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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 ー 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

- 「LONG VARBIT データ型」 65 ページ
- 「ビット配列の変換」 85 ページ
- 「ビット配列関数」 94 ページ
- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

日付と時刻データ型

ここでは、SQL Anywhere 内での日付の値の処理方法や、年を表す 2 桁の文字列値の変換などのあいまいな日付情報の処理方法について説明します。

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は常に、年の値の世紀にあたる部分を明示的にインクルードします。
- 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。

日付の格納方法

年の値を含む日付は、次のいずれかのデータ型を使用して、SQL Anywhere データベースの内部で使用され格納されます。

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 ー 9999-12-31
TIMESTAMP	タイムスタンプ (年、月、日、時、分、秒、秒以下 (6 桁まで))	8 バイト	0001-01-01 ー 9999-12-31 (TIMESTAMP の時間の精度については、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る)

SQL Anywhere の日付と時刻データ型の詳細については、「日付と時刻データ型」 67 ページを参照してください。

日付と時刻をデータベースに送信する

次のいずれかの方法で、日付と時刻をデータベースに送信します。

- インタフェースを使うときは、文字列として
- ODBC を使うときは、TIMESTAMP 構造体として
- Embedded SQL を使うときは、SQLDATETIME 構造体として

時刻をデータベースに文字列 (TIME データ型用) または文字列の一部 (TIMESTAMP データ型または DATE データ型用) として送信する場合、時間、分、秒をコロンで区切って *hh*:*mm*:*ss.sss* フォーマットにする必要はありますが、文字列中のどこにでも置けます。次に、時刻を指定する場合の有効で明瞭な文字列を示します。

```
21:35 -- 24 hour clock if no am or pm specified
10:00pm -- pm specified, so interpreted as 12 hour clock
10:00 -- 10:00am in the absence of pm
10:23:32.234 -- seconds and fractions of a second included
```

データベースに日付を文字列として送信すると、文字列は自動的に DATE と TIMESTAMP のデータ型に変換されます。文字列は、次の 2 つの方法のいずれかで送信します。

- データベースに確実に解釈される *yyyy/mm/dd* または *yyyy-mm-dd* のいずれかの文字列として。
- date_order データベース・オプションに従って解釈される文字列として。「date_order オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Transact-SQL の文字列から日付／時刻への変換

文字列から日付と時刻のデータ型への変換

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。

例

SQL Anywhere では、セパレータ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 to 12:34:56.700
12:34:56:7 to 12:34:56.700
12.34.56.78 to 12:34:56.780
12.34.56:78 to 12:34:56.780
12:34:56.789 to 12:34:56.789
12:34:56:789 to 12:34:56.789
```

データベースから日付と時刻を取得する

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します。

- インタフェースを使うときは、文字列として
- ODBC を使うときは、TIMESTAMP 構造体として
- Embedded SQL を使うときは、SQLDATETIME 構造体として

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 によって指定されているフォーマットで検索します。これらのオプションの説明については、「SET OPTION 文」 686 ページを参照してください。

日付と時刻を扱う関数の詳細については、「日付と時刻関数」 95 ページを参照してください。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 "Tuesday" 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。

日付と時刻を比較する

デフォルトでは、DATE として格納された値には時または分の値が含まれていないため、日付の比較は単純です。

DATE データ型に時刻を指定することもできます。ただし、日付と比較するときに複雑になります。日付をデータベースに入力するときに時刻を指定しない場合、時刻のデフォルトは 0:00 または 12:00am (真夜中) になります。このオプションを設定した場合、日付の比較には必ず日付とともに時刻も含めます。データベースの日付値 1999-05-23 10:00 は、定数 1999-05-23 と等価ではありません。DATEFORMAT 関数または他の日付関数を使用して日付と時刻のフィールドの各部を比較することができます。次に例を示します。

```
DATEFORMAT(invoice_date,'yyyy/mm/dd') = '1999/05/23'
```


データベース・カラムに日付だけが必要な場合、クライアント・アプリケーション側で、データベースへデータを入力するときに時刻を指定しないようにしてください。このようにすると、日付だけの文字列による比較が目的どおりに行われます。

日付を*文字列として*比較する場合は、DATEFORMAT 関数または CAST 関数を使って日付を文字列に変換してから比較してください。

あいまいさのない日付と時刻の使用

yyyy/mm/dd または *yyyy-mm-dd* フォーマットの日付は、date_order 設定がされているかどうかにかかわらず、日付として常に正確に認識されます。セパレータとして "**/**" または "**-**" を使う代わりに、他の文字を使うこともできます (たとえば、"**?**"、スペース、"**,**" など)。別のユーザが異なる date_order 設定をしているコンテキストの中では、このフォーマットを使うようにしてください。たとえば、ストアド・プロシージャでは、あいまいさのない日付フォーマットを使えば、ユーザの date_order 設定に従った日付が間違って解釈されることはありません。

また、*hh:mm:ss:ssss* の形式の文字列は、あいまいさのない時刻として解釈されます。

日付と時刻を組み合わせる場合は、あいまいさのない日付と時刻を組み合わせることにより、結果的にあいまいさのない日付／時刻として評価されます。*yyyy-mm-dd hh.mm.ss.sss* の形式も指定できます。

あいまいさのない日付／時刻値です。ピリオドは、日付と組み合わせた時刻に対してのみ使用できます。

他のコンテキストでは、より柔軟性のある日付フォーマットを使用できます。SQL Anywhere は、広い範囲の文字列を日付として解釈します。この解釈は、date_order データベース・オプションに左右されます。date_order データベース・オプションは、値に *MDY*、*YMD*、*DMY* を取ります (「SET OPTION 文」 686 ページを参照してください)。たとえば、次の文は、date_order オプションを *DMY* に設定します。

```
SET OPTION date_order = 'DMY' ;
```

デフォルトの date_order 設定は YMD です。ODBC ドライバは、接続されるときに必ず date_order オプションを YMD に設定します。値は SET TEMPORARY OPTION 文を使って変更できます。

データベース・オプション date_order は、文字列 10/11/12 がデータベースで 2010 年 11 月 12 日、2012 年 10 月 11 日、または 2012 年 11 月 10 日のいずれとして解釈されるのかを決定します。日付文字列の年、月、日を記号 (たとえば /、-、またはスペース) で区切り、date_order オプションで指定されている順序で表してください。

年は 2 桁または 4 桁で指定できます。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 未満の値には 2000 が追加され、その他のすべての値には 1900 が追加されます。このオプションのデフォルト値は 50 です。したがって、デフォルトで 50 は 1950、49 は 2049 と解釈されます。

月は月の名前または数字です。時間と分はコロンで区切りますが、文字列のどこにでも置けます。

注意

- 年は常に 4 桁フォーマットで指定することをおすすめします。
- date_order の適切な設定を使った次の文字列は、すべて有効な日付です。

  99-05-23 21:35
  99/5/23
  1999/05/23
  May 23 1999
  23-May-1999
  Tuesday May 23, 1999 10:00pm

- 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**　今年
  - **月**　デフォルトなし
  - **日**　1 (月のフィールドに対して便利です。たとえば、1999 年 5 月は、日付 1999-05-01 00:00 となります)
  - **時、分、秒、秒以下**　0

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

年の値の範囲は 0001 ー 9999 です。SQL Anywhere の最小日付は 0001-01-01 00:00:00 です。

従来の動作を考慮して、DATE カラムには時刻と分も指定することができます。時と分を含むデータには、TIMESTAMP データ型の使用をおすすめします。

アプリケーションが DATE 値を取得するときのフォーマットは、date_format の設定によって決まります。たとえば、2003 年 7 月 19 日を表す日付値は、2003/07/19、Jul 19, 2003 など、さまざまなフォーマットで返されます。

データベース・サーバが文字列を解釈する方法は、date_order オプションによって決まります。たとえば、アプリケーションが DATE 値として渡した値 02/05/2002 は、date_order オプションの設定によって、5 月 2 日または 2 月 5 日と解釈されます。

DATE 値には 4 バイトの記憶領域が必要です。

参照

- 「date_format オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「date_order オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「DATETIME データ型」 72 ページ
- 「SMALLDATETIME データ型」 72 ページ

- 「TIMESTAMP データ型」 73 ページ
- 「日付と時刻関数」 95 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

DATETIME データ型

DATETIME データ型は、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。

構文

DATETIME

参照

- 「DATE データ型」 71 ページ
- 「SMALLDATETIME データ型」 72 ページ
- 「TIMESTAMP データ型」 73 ページ
- 「日付と時刻関数」 95 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

SMALLDATETIME データ型

SMALLDATETIME データ型は、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。

構文

SMALLDATETIME

参照

- 「DATE データ型」 71 ページ
- 「DATETIME データ型」 72 ページ
- 「TIMESTAMP データ型」 73 ページ
- 「日付と時刻関数」 95 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

秒以下は 6 桁まで格納されます。TIME 値には 8 バイトの記憶領域が必要です(ODBC 規格では、TIME データ型の精度を秒単位までに制限しています)。このため、WHERE 句の比較に、秒の単位より高い精度に依存する TIME データ型を使用しないでください)。

参照

- 「TIMESTAMP データ型」 73 ページ
- 「日付と時刻関数」 95 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

備考

秒以下は 6 桁まで格納されます。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 ー 9999 年の日付を格納でき、さらに 1600-02-28 23:59:59 ー 7911-01-01 00:00:00 も格納できます。TIMESTAMP では、この範囲より前または後の時刻を正しく表せない場合があります。

参照

- 「TIME データ型」 72 ページ
- 「日付と時刻関数」 95 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

max-length　値のバイト単位での最大長。長さの範囲は 1 ー 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」 76 ページ
- 「LONG BINARY データ型」 75 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 のドメインとして実装されます。

参照

- 「LONG BINARY データ型」 75 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

最大値は 2 GB です。

参照

- 「BINARY データ型」 74 ページ
- 「VARBINARY データ型」 76 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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、あるコンピュータで生成される UUID 値が他のコンピュータで生成される 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。デフォルトで、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 - SQL の使用法』
- 「NEWID 関数 [その他]」 206 ページ
- 「UUIDTOSTR 関数 [文字列]」 276 ページ
- 「STRTOUUID 関数 [文字列]」 263 ページ
- 「uuid_has_hyphens オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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* **)**]

パラメータ

max-length 値のバイト単位での最大長。長さの範囲は 1 ー 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」 74 ページ
- 「LONG BINARY データ型」 75 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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。構文の詳細については、「CREATE DOMAIN 文」 392 ページを参照してください。

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 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だけです。使用中のドメインを削除しようとすると、「テーブル 'SYSUSERTYPE' のプライマリ・キーは他のテーブルから参照されています。」とメッセージが表示されます。

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 );
```

データ型 **id** を使って作成されたカラムは、NULL 値を保持できないので、デフォルトで値を自動的に増やし、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提供すると、必要に応じて、データ型の属性を上書きできます。データ型 **id** 上で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。
- **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。そして、変換エラーをレポートします。
- **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」 94 ページ
- 「DATE 関数 [日付と時刻]」 137 ページ
- 「STRING 関数 [文字列]」 262 ページ
- 「CAST 関数 [データ型変換]」 115 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。「CAST 関数 [データ型変換]」 115 ページを参照してください。

置換文字

変換後の文字セットで文字を表せない場合は、代わりに置換文字セットが使用されます。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せない場合、その文字が失われます。

また、さまざまな文字セットに対して取り得る置換文字が複数あるだけでなく、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可能性があります。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、非ユニコードのクライアント・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 UTF-16 値にフェッチされる (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 UTF-16 文字 0x001A になります。しかし、これはユーロ記号として UTF-16 に定義された置換文字ではありません。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に定義された文字と一致しない可能性があることを示しています。

そのため、複数の文字セット間で変換を行うときに置換文字が使用される方法を理解してテストすることが重要です。

on_charset_conversion_failure オプションは、文字が変換後の文字セットで表せない場合の変換時の動作を指定します。「on_charset_conversion_failure オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

参照

- 「データ型変換」 80 ページ
- 「CHAR と NCHAR の比較」 81 ページ
- 「on_charset_conversion_failure オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』

CHAR と NCHAR の比較

CHAR 型 (CHAR、VARCHAR、LONG VARCHAR) と NCHAR 型 (NCHAR、NVARCHAR、LONG NVARCHAR) との間で比較を実行する場合、NCHAR 値を CHAR 型に**強制**できるか (さらには強制するべきか) どうかを判断するために推定規則が使用されます。値がリテラル定数、変数、またはホスト変数であるか、カラム参照を基にしない複雑な式である場合は、値を強制できます。一般に、NCHAR 値を CHAR カラムと比較する場合、NCHAR 値を CHAR に強制できれば CHAR として、そうでなければ NCHAR として、比較が実行されます。

次に、推定規則を適用順に示します。

- 強制不可能な NCHAR 値が存在する場合、すべての CHAR 値が NCHAR に変換され、NCHAR として比較が行われる。
- または、強制不可能な CHAR 値が存在する場合、すべての NCHAR 値が CHAR に変換され、CHAR として比較が行われる。

  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。詳細については、「NCHAR から CHAR への変換」 84 ページを参照してください。
- または、強制可能な CHAR 値と NCHAR 値が混在する場合 (つまりすべての値が強制可能)、すべての CHAR 値が NCHAR に変換され、NCHAR として比較が行われる。

例

条件 Employees.GivenName = N'Susan' 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 に強制されるため、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 が CHAR に強制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」 84 ページ
- 「置換文字」 81 ページ
- 「CAST 関数 [データ型変換]」 115 ページ
- 「CONVERT 関数 [データ型変換]」 125 ページ
- 「CAST 関数 [データ型変換]」 115 ページ
- 「on_charset_conversion_failure オプション [データベース]」 『SQL Anywhere サーバ - 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。

日時データ型間の比較

SQL Anywhere では、日時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. いずれかの引数のデータ型が TIME の場合は、両方を TIME に変換してから比較を行う。
2. いずれかのデータ型が DATE または TIMESTAMP の場合は、両方を TIMESTAMP 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 DATE 型の場合、これらの引数は TIMESTAMP に変換されます。
3. 一方の引数が NUMERIC 型で、もう一方の引数が FLOAT 型の場合は、両方を DOUBLE に変換してから比較を行う。

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。「CHAR と NCHAR の比較」 81 ページを参照してください。

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。

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。この種の変換は、**損失を伴う**と考えられます。CHAR 型で表せない NCHAR 文字があるからです。このような文字が NCHAR データに含まれる場合、CHAR 文字セットの置換文字が代わりに使用されます。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。「on_charset_conversion_failure オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

参照

- 「CHAR と NCHAR の比較」 81 ページ
- 「on_charset_conversion_failure オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

NULL 定数を NUMERIC 型と文字列型に変換する

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

日付から文字列への変換

SQL Anywhere には、SQL Anywhere の日付と時刻の値をさまざまな文字列や式に変換するための機能が数多く用意されています。データ値を文字列に変換する場合は、年の最初の 2 桁を削除して残りの 2 桁で年を表すことができます。

間違った世紀の値

次の文は、正しく文字列に変換できなかった例です。データベース・エラーは発生しませんが、日付が January 1 2000 ではなく January 1, 1900 という日付を表す文字列に変換されています。

```
SELECT DATEFORMAT (
        DATEFORMAT('2000-01-01', 'Mmm dd/yy' ),
        'yyyy-Mmm-dd' )
   AS Wrong_year;
```

SQL Anywhere は自動的にあいまいな日付文字列 2000-01-01 を正しい日付値に変換します。ただし、内部またはネストされた DATEFORMAT 関数の 'Mmmdd/yy' フォーマットは、文字列に再び変換され、外部の DATEFORMAT 関数に渡されるときに最初の 2 桁が削除されます。

この場合、データベース・オプション nearest_century が 0 に設定されているため、外部の DATEFORMAT 関数は、2 桁で年を表す文字列を 1900 ー 1999 年の間の年を表す値に変換します。

日付と時刻関数の詳細については、「日付と時刻関数」 95 ページを参照してください。

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、ビット配列の値は整数の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) 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。

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ゼロが使用されます。

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 '01110' の文字列を返します。

数値セット間の変換

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]」 115 ページ
- 「CONVERT 関数 [データ型変換]」 125 ページ
- 「CAST 関数 [データ型変換]」 115 ページ

あいまいな文字列から日付への変換

SQL Anywhere では、期待値が日付の値であれば、年が文字列内で 2 桁だけで表されている場合でも、文字列を日付に自動的に変換します。

年の値の世紀を表す部分が省略されている場合は、nearest_century データベース・オプションによって変換方法が決まります。

nearest_century データベース・オプションは、日付値 19YY と日付値 20YY の間を区切る数値です。

nearest_century 値よりも小さい 2 桁の年は 20yy に変換され、nearest_century 値以上の年は 19yy に変換されます。

このオプションを設定しないと、デフォルト設定の 50 が使用されます。したがって、2 桁の年文字列は 1950 ー 2049 として認識されます。

この nearest_century オプションは、SQL Anywhere バージョン 5.5 から導入されました。バージョン 5.5 では、デフォルト設定は 0 でした。

あいまいな日付の変換例

次の文は、SQL Anywhere でのあいまいな日付情報の変換を表すのに使用されるテーブルを作成します。

```
CREATE TABLE T1 (C1 DATE);
```

table T1 は、DATE 型の 1 カラム C1 を含みます。

次の文は、カラム C1 に日付の値を挿入します。SQL Anywhere ではあいまいな年の値を含む文字列は、年だけを表して世紀を表さない 2 桁の値に自動的に変換されます。

```
INSERT INTO T1 VALUES('00-01-01');
```

デフォルトでは、nearest_century オプションは 50 に設定されているので、SQL Anywhere では、文字列を 2000-01-01 という日付に変換します。次の文は、この挿入の結果を確認します。

```
SELECT * FROM T1;
```

次の文を使用して nearest_century オプションを変更すると、変換処理が変更されます。

```
SET OPTION nearest_century = 0;
```

nearest_century オプションが 0 に設定されている場合、同じ文を使用して直前の挿入を実行すると、異なる日付の値が作成されます。

```
INSERT INTO T1 VALUES('00-01-01');
```

上の文を実行すると、1900-01-01 という日付が挿入されます。次の文を使用して結果を確認します。

```
SELECT * FROM T1;
```


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	INTEGER
float	REAL
double	DOUBLE
byte[ ]	VARBINARY
byte[ ]	IMAGE
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.lang.Double	DOUBLE
java.lang.Float	REAL
java.lang.Integer	INTEGER

Java 型	SQL 型
java.lang.Long	INTEGER
void	this[1]

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
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[ ]
VARBINARY	byte[ ]
LONG VARBINARY	byte[ ]
IMAGE	byte[ ]
DATE	java.sql.Date

1 このメソッドはオブジェクトそのものを返します。

SQL 型	Java 型
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

第 3 章

SQL 関数

目次

SQL 関数の概要

関数は、データベースの情報を返すために使用します。式を使用できるところでは必ず関数を使用できます。

関数には、SQL 文と同じ構文の表記規則を使用します。構文の表記規則の全リストについては、「SQL 構文の表記規則」 299 ページを参照してください。

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。

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]」 107 ページ
- 「BIT_AND 関数 [集合]」 110 ページ
- 「BIT_OR 関数 [集合]」 111 ページ
- 「BIT_XOR 関数 [集合]」 112 ページ
- 「COVAR_POP 関数 [集合]」 131 ページ
- 「COVAR_SAMP 関数 [集合]」 132 ページ
- 「COUNT 関数 [集合]」 130 ページ
- 「CORR 関数 [集合]」 127 ページ
- 「FIRST_VALUE 関数 [集合]」 166 ページ
- 「GROUPING 関数 [集合]」 173 ページ
- 「LAST_VALUE 関数 [集合]」 189 ページ
- 「LIST 関数 [集合]」 193 ページ
- 「MAX 関数 [集合]」 199 ページ
- 「MIN 関数 [集合]」 200 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_R2 関数 [集合]」 228 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「SET_BITS 関数 [集合]」 246 ページ
- 「STDDEV 関数 [集合]」 258 ページ
- 「STDDEV_POP 関数 [集合]」 259 ページ
- 「STDDEV_SAMP 関数 [集合]」 260 ページ
- 「SUM 関数 [集合]」 266 ページ
- 「VAR_POP 関数 [集合]」 277 ページ
- 「VAR_SAMP 関数 [集合]」 278 ページ
- 「VARIANCE 関数 [集合]」 280 ページ
- 「XMLAGG 関数 [集合]」 282 ページ

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]」 110 ページ
- 「BIT_OR 関数 [集合]」 111 ページ
- 「BIT_XOR 関数 [集合]」 112 ページ
- 「BIT_LENGTH 関数 [ビット配列]」 109 ページ
- 「BIT_SUBSTR 関数 [ビット配列]」 109 ページ
- 「COUNT_SET_BITS 関数 [ビット配列]」 131 ページ
- 「GET_BIT 関数 [ビット配列]」 168 ページ
- 「SET_BIT 関数 [ビット配列]」 245 ページ
- 「SET_BITS 関数 [集合]」 246 ページ

ビット処理演算子の詳細については、「ビット処理演算子」 13 ページを参照してください。

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]」 135 ページ
- 「DENSE_RANK 関数 [ランキング]」 152 ページ
- 「PERCENT_RANK 関数 [ランキング]」 214 ページ
- 「RANK 関数 [ランキング]」 222 ページ

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]」 115 ページ
- 「CONVERT 関数 [データ型変換]」 125 ページ
- 「HEXTOINT 関数 [データ型変換]」 175 ページ
- 「INTTOHEX 関数 [データ型変換]」 186 ページ
- 「ISDATE 関数 [データ型変換]」 186 ページ
- 「ISNUMERIC 関数 [その他]」 188 ページ

日付と時刻関数

日付と時刻関数は、date データ型と time データ型の操作を行ったり、日付または時刻の情報を返したりします。

この章では、「**日時**」を日付、時刻、またはタイムスタンプを意味する用語として使用しています。特定のデータ型 DATETIME を示す場合は、DATETIME を使用します。

DATETIME データ型の詳細については、「日付と時刻データ型」 67 ページを参照してください。

関数のリスト

次の日付と時刻関数を使用できます。

日付の単位

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。

日付の単位	省略形	値の範囲
Year	yy	1–9999
Quarter	qq	1–4
Month	mm	1–12
Week	wk	1 ー 54。日曜日を週の最初の日とします。
Day	dd	1–31
Dayofyear	dy	1–366
Weekday	dw	1ー 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
Calyearofweek	cyr	整数。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ー木曜日の場合は、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ー日曜日の場合、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 ー 54。指定した日付がその年の第何週であるかを示します。
Caldayofweek	cdw	1 ー 7 (月曜日 = 1、…、日曜日 = 7)

ユーザ定義関数

SQL Anywhere には、ユーザ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。

SQL のユーザ定義関数

「CREATE FUNCTION 文」 408 ページを使用して、独自の SQL 関数を実装できます。CREATEFUNCTION 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。

SQL 関数の作成の詳細については、「プロシージャ、トリガ、バッチの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

Java のユーザ定義関数

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、標準のユーザ定義関数とは動作が異なります。

Java クラス作成とクラス・メソッドの詳細については、「クラスの作成」『SQL Anywhere サーバ - プログラミング』を参照してください。

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]」 104 ページ
- 「COALESCE 関数 [その他]」 118 ページ
- 「COMPRESS 関数 [文字列]」 121 ページ
- 「CONFLICT 関数 [その他]」 123 ページ
- 「DECOMPRESS 関数 [文字列]」 149 ページ
- 「DECRYPT 関数 [文字列]」 150 ページ
- 「ENCRYPT 関数 [文字列]」 154 ページ
- 「ERRORMSG 関数 [その他]」 156 ページ
- 「ESTIMATE 関数 [その他]」 157 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」 157 ページ
- 「EXPERIENCE_ESTIMATE 関数 [その他]」 163 ページ
- 「EXPLANATION 関数 [その他]」 164 ページ
- 「EXPRTYPE 関数 [その他]」 165 ページ
- 「GET_IDENTITY 関数 [その他]」 169 ページ
- 「GRAPHICAL_PLAN 関数 [その他]」 171 ページ
- 「GREATER 関数 [その他]」 172 ページ
- 「IDENTITY 関数 [その他]」 183 ページ
- 「IFNULL 関数 [その他]」 184 ページ
- 「INDEX_ESTIMATE 関数 [その他]」 185 ページ
- 「ISNULL 関数 [その他]」 187 ページ
- 「LESSER 関数 [その他]」 193 ページ
- 「NEWID 関数 [その他]」 206 ページ

- 「NULLIF 関数 [その他]」 212 ページ
- 「NUMBER 関数 [その他]」 212 ページ
- 「PLAN 関数 [その他]」 216 ページ
- 「REWRITE 関数 [その他]」 238 ページ
- 「ROW_NUMBER 関数 [その他]」 242 ページ
- 「SQLDIALECT 関数 [その他]」 256 ページ
- 「SQLFLAGGER 関数 [その他]」 257 ページ
- 「TRACEBACK 関数 [その他]」 270 ページ
- 「TRANSACTSQL 関数 [その他]」 271 ページ
- 「VAREXISTS 関数 [その他]」 280 ページ
- 「WATCOMSQL 関数 [その他]」 280 ページ

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。

関数のリスト

次の数値関数を使用できます。

- 「ABS 関数 [数値]」 103 ページ
- 「ACOS 関数 [数値]」 103 ページ
- 「ASIN 関数 [数値]」 105 ページ
- 「ATAN 関数 [数値]」 106 ページ
- 「ATAN2 関数 [数値]」 106 ページ
- 「CEILING 関数 [数値]」 115 ページ
- 「COS 関数 [数値]」 128 ページ
- 「COT 関数 [数値]」 129 ページ
- 「DEGREES 関数 [数値]」 151 ページ
- 「EXP 関数 [数値]」 163 ページ
- 「FLOOR 関数 [数値]」 168 ページ
- 「LOG 関数 [数値]」 196 ページ
- 「LOG10 関数 [数値]」 197 ページ
- 「MOD 関数 [数値]」 202 ページ
- 「PI 関数 [数値]」 215 ページ
- 「POWER 関数 [数値]」 217 ページ
- 「RADIANS 関数 [数値]」 220 ページ
- 「RAND 関数 [数値]」 221 ページ
- 「REMAINDER 関数 [数値]」 234 ページ
- 「ROUND 関数 [数値]」 240 ページ
- 「SIGN 関数 [数値]」 248 ページ
- 「SIN 関数 [数値]」 249 ページ
- 「SQRT 関数 [数値]」 258 ページ
- 「TAN 関数 [数値]」 267 ページ
- 「TRUNCNUM 関数 [数値]」 272 ページ

HTTP 関数と SOAP 関数

HTTP 関数は、Web サービス内の HTTP 要求処理を容易にします。同様に、SOAP 関数は、Web サービス内の SOAP 要求処理を容易にします。

関数のリスト

次の HTTP 関数を使用できます。

- 「HTML_DECODE 関数 [その他]」 178 ページ
- 「HTML_ENCODE 関数 [その他]」 178 ページ
- 「HTTP_DECODE 関数 [HTTP]」 179 ページ
- 「HTTP_ENCODE 関数 [HTTP]」 180 ページ
- 「HTTP_HEADER 関数 [HTTP]」 181 ページ
- 「HTTP_VARIABLE 関数 [HTTP]」 182 ページ
- 「NEXT_HTTP_HEADER 関数 [HTTP]」 209 ページ
- 「NEXT_HTTP_VARIABLE 関数 [HTTP]」 210 ページ

次の SOAP 関数を使用できます。

- 「NEXT_SOAP_HEADER 関数 [SOAP]」 210 ページ
- 「SOAP_HEADER 関数 [SOAP]」 250 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。

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]」 104 ページ
- 「BASE64_DECODE 関数 [文字列]」 108 ページ
- 「BASE64_ENCODE 関数 [文字列]」 108 ページ
- 「BYTE_LENGTH 関数 [文字列]」 113 ページ
- 「BYTE_SUBSTR 関数 [文字列]」 114 ページ
- 「CHAR 関数 [文字列]」 116 ページ
- 「CHARINDEX 関数 [文字列]」 117 ページ
- 「CHAR_LENGTH 関数 [文字列]」 118 ページ
- 「COMPARE 関数 [文字列]」 119 ページ
- 「COMPRESS 関数 [文字列]」 121 ページ
- 「CSCONVERT 関数 [文字列]」 133 ページ
- 「DECOMPRESS 関数 [文字列]」 149 ページ
- 「DECRYPT 関数 [文字列]」 150 ページ

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]」 122 ページ
- 「CONNECTION_PROPERTY 関数 [システム]」 123 ページ
- 「DATALENGTH 関数 [システム]」 136 ページ
- 「DB_ID 関数 [システム]」 147 ページ
- 「DB_NAME 関数 [システム]」 147 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」 144 ページ
- 「DB_PROPERTY 関数 [システム]」 148 ページ
- 「EVENT_CONDITION 関数 [システム]」 159 ページ
- 「EVENT_CONDITION_NAME 関数 [システム]」 160 ページ
- 「EVENT_PARAMETER 関数 [システム]」 161 ページ
- 「NEXT_CONNECTION 関数 [システム]」 207 ページ
- 「NEXT_DATABASE 関数 [システム]」 208 ページ
- 「PROPERTY 関数 [システム]」 217 ページ
- 「PROPERTY_DESCRIPTION 関数 [システム]」 218 ページ
- 「PROPERTY_NAME 関数 [システム]」 219 ページ
- 「PROPERTY_NUMBER 関数 [システム]」 219 ページ

注意

- 一部のシステム関数は、ストアド・プロシージャとして 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。

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
datalength(*expression* **)**	式の長さを返す (バイト)
db_id([*database-name*] **)**	データベース ID 番号を返す
db_name([*database-id*] **)**	データベースの名前を返す
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
suser_id([*user-name*] **)**	整数のユーザ ID 番号を返す
suser_name([*user-id*] **)**	ユーザ ID を返す

システム関数	説明
tsequal (*timestamp*, *timestamp2* **)**	SQL Anywhere では、tsequal は、タイムスタンプ値 (ミリ秒にトランケートされた値) を比較して、選択後に修正されたローの更新を防止する。タイムスタンプが異なる場合、false (0) が返される。「更新時の tsequal の使用」『SQL Anywhere サーバ - SQL の使用法』を参照。tsequal は使用されなくなった。
user_id([*user-name*] **)**	整数のユーザ ID 番号を返す。SQL Anywhere のユーザ ID は返さない。
user_name([*user-id*] **)**	ユーザ ID 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]」 267 ページ

アルファベット順の関数リスト

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」 93 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

numeric-expression　絶対値が返される数値。

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能。

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で返します。

構文

ACOS(*numeric-expression* **)**

パラメータ

numeric-expression　角度のコサイン。

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。

参照

- 「ASIN 関数 [数値]」 105 ページ
- 「ATAN 関数 [数値]」 106 ページ
- 「ATAN2 関数 [数値]」 106 ページ
- 「COS 関数 [数値]」 128 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。

備考

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。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/2003**　ベンダ拡張。

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

string-expression　文字列。

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

- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 90 を返します。

```
SELECT ASCII( 'Z' );
```

ASIN 関数 [数値]

数値式のアークサインをラジアンで返します。

構文

ASIN(*numeric-expression* **)**

パラメータ

numeric-expression 角度のサイン。

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、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

- 「ACOS 関数 [数値]」 103 ページ
- 「ATAN 関数 [数値]」 106 ページ
- 「ATAN2 関数 [数値]」 106 ページ
- 「SIN 関数 [数値]」 249 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

数値式のアークタンジェントをラジアンで返します。

構文

ATAN(*numeric-expression* **)**

備考

ATAN 関数と TAN 関数は逆変換の演算です。

パラメータ

numeric-expression　角度のタンジェント。

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。

参照

- 「ACOS 関数 [数値]」 103 ページ
- 「ASIN 関数 [数値]」 105 ページ
- 「ATAN2 関数 [数値]」 106 ページ
- 「TAN 関数 [数値]」 267 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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で返します。

構文

{ ATN2 | ATAN2 }(*numeric-expression-1*, *numeric-expression-2* **)**

パラメータ

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。

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。

参照

- 「ACOS 関数 [数値]」 103 ページ
- 「ASIN 関数 [数値]」 105 ページ
- 「ATAN 関数 [数値]」 106 ページ
- 「TAN 関数 [数値]」 267 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、比率 0.52 ー 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、数値式または設定されたユニークな値の平均値を計算します。

構文 1

AVG(*numeric-expression* | **DISTINCT** *numeric-expression* **)**

構文 2

AVG(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec : 下の「備考」の構文 2 の説明を参照

パラメータ

numeric-expression　ロー・セットで平均値を計算する対象の式。

DISTINCT 数値式　入力中で一意の数値の平均を計算します。

備考

この平均値には、*numeric-expression* が NULL 値のローは含まれません。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。

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」 745 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 - SQL の使用法』を参照してください。

参照

- 「SUM 関数 [集合]」 266 ページ
- 「COUNT 関数 [集合]」 130 ページ

標準と互換性

- **SQL/2003**　コア機能。構文 2 は機能 T611 です。

例

次の文は、値 49988.623200 を返します。

```
SELECT AVG( Salary ) FROM Employees ;
```

次の文は、製品リストの一意の価格に基づいて平均値を計算するときに使用できます。

```
SELECT AVG( DISTINCT ListPrice ) FROM Production ;
```

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

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。

参照

- 「BASE64_ENCODE 関数 [文字列]」 108 ページ
- 「文字列関数」 99 ページ

標準と互換性

SQL/2003　ベンダ拡張。

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

string-expression　エンコードされる文字列。

参照

- 「BASE64_DECODE 関数 [文字列]」 108 ページ
- 「文字列関数」 99 ページ

標準と互換性

SQL/2003 ベンダ拡張。

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_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

bit-expression 長さを決定するビット式。

参照

- 「BIT_LENGTH 関数 [ビット配列]」 109 ページ
- 「CHAR_LENGTH 関数 [文字列]」 118 ページ

標準と互換性

SQL/2003 ベンダ拡張。

例

次の文は、値 8 を返します。

```
SELECT BIT_LENGTH( '01101011' );
```

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

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。

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、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。

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]」 264 ページ

標準と互換性

SQL/2003　ベンダ拡張。

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。

```
SELECT BIT_SUBSTR( '01011011101111011111', 2, 5 );
```

次の文は、11111 を返します。

```
SELECT BIT_SUBSTR( '01011011101111011111', -5, 5 );
```

BIT_AND 関数 [集合]

n ビット配列を取得し、引数をビット処理で AND 演算した値を返します。このとき各ビットを比較してすべてのビットが 1 の場合は 1 を返し、それ以外の場合は 0 を返します。

構文

BIT_AND(*bit-expression* **)**

パラメータ

expression　値を計算する式。通常はカラム名です。

参照

- 「BIT_OR 関数 [集合]」 111 ページ
- 「BIT_XOR 関数 [集合]」 112 ページ
- 「ビット処理演算子」 13 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

データ型が VARBIT カラムを 1 つ含む次の表 t があるとします。

a
0001
0111
0100
0011

たとえば、カラムの AND 値を調べる場合、次の SELECT 文を入力すると、0000 が返されます。

```
SELECT BIT_AND( a ) FROM t;
```

結果は次のように決定されます。

1. Row 1 (0001) と Row 2 (0111) が比較され、結果は 0001 になります (どちらの値も 4 番目のビットに 1 があります)。
2. 前の比較結果 (0001) と Row 3 (0100) が比較され、結果は 0000 になります (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 Row 4 (0011) が比較され、結果は 0000 になります (同じビット位置に 1 がありませんでした)。

BIT_OR 関数 [集合]

n ビット配列を取得し、引数をビット処理で OR 演算した値を返します。このとき各ビットを比較して任意のビットが 1 の場合は 1 を返し、それ以外の場合は 0 を返します。

構文

BIT_OR(*bit-expression* **)**

パラメータ

expression　値を計算する式。通常はカラム名です。

参照

- 「BIT_AND 関数 [集合]」 110 ページ
- 「BIT_XOR 関数 [集合]」 112 ページ
- 「ビット処理演算子」 13 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

データ型が VARBIT カラムを 1 つ含む次の表 t があるとします。

a
0001
0111
0100
0011

たとえば、カラムの OR 値を調べる場合、次の SELECT 文を入力すると、0111 が返されます。

```
SELECT BIT_OR( a ) FROM t;
```

結果は次のように決定されます。

1. Row 1 (0001) と Row 2 (0111) が比較され、結果は 0111 になります。
2. 前の比較結果 (0111) と Row 3 (0100) が比較され、結果は 0111 になります。
3. 前の比較結果 (0111) と Row 4 (0011) が比較され、結果は 0111 になります。

BIT_XOR 関数 [集合]

n ビット配列を取得し、引数をビット処理で排他的 OR 演算した値を返します。各ビットを比較し、設定ビットに奇数個の引数がある場合 (奇数パリティ) は 1 を返し、それ以外の場合は 0 を返します。

構文

BIT_XOR(*bit-expression* **)**

パラメータ

expression　値を計算する式。通常はカラム名です。

参照

- 「BIT_AND 関数 [集合]」 110 ページ
- 「BIT_OR 関数 [集合]」 111 ページ
- 「ビット処理演算子」 13 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

データ型が VARBIT カラムを 1 つ含む次の表 t があるとします。

a
0001
0111
0100
0011

たとえば、カラムの XOR 値を調べる場合、次の SELECT 文を入力すると、0001 が返されます。

```
SELECT BIT_XOR( a ) FROM t;
```

結果は次のように決定されます。

1. Row 1 (0001) と Row 2 (0111) が比較され、結果は 0110 になります。
2. 前の比較結果 (0110) と Row 3 (0100) が比較され、結果は 0010 になります。
3. 前の比較結果 (0010) と Row 4 (0011) が比較され、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

string-expression　長さが計算される文字列。

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]」 118 ページ

- 「DATALENGTH 関数 [システム]」 136 ページ
- 「LENGTH 関数 [文字列]」 192 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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*] **)**

パラメータ

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、取得するバイト数が開始位置から*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。

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。

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]」 264 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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。

構文

CAST(*expression* **AS** *datatype* **)**

パラメータ

expression　変換される式。

data type　ターゲットのデータ型。

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

- 「CONVERT 関数 [データ型変換]」 125 ページ
- 「LEFT 関数 [文字列]」 191 ページ

標準と互換性

- **SQL/2003**　コア機能。

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]

数値の切り上げ値を返します。

構文

CEILING(*numeric-expression* **)**

パラメータ

numeric-expression　切り上げ値を計算する対象の数値。

備考

Ceiling 関数は指定した値以上の最初の整数を返します。正の数値の場合、「丸め」とも呼ばれます。

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。

参照

- 「FLOOR 関数 [数値]」 168 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

integer-expression　ASCII 文字に変換される数値。0 ー 255 の範囲内の数を指定します。

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」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 Y を返します。

```
SELECT CHAR( 89 );
```

CHARINDEX 関数 [文字列]

ある文字列内でのもう 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

string-expression-1　検索する文字列。

string-expression-2　検索される文字列。

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。

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]」 264 ページ
- 「REPLACE 関数 [文字列]」 236 ページ
- 「LOCATE 関数 [文字列]」 195 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、名に K という文字が含まれる場合にのみ、Surname テーブルと GivenName テーブル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

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

string-expression　長さが計算される文字列。

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]」 113 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の文は、値 8 を返します。

```
SELECT CHAR_LENGTH( 'Chemical' );
```

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

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となっている必要があります。

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。

この関数がデータベース・サーバで処理される方法の詳細については、「ISNULL 関数 [その他]」 187 ページを参照してください。

参照

- 「ISNULL 関数 [その他]」 187 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の文は、値 34 を返します。

```
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

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]」 251 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・ユニコード・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]」 251 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。「照合の適合化オプション」 382 ページを参照してください。

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。

備考

選択した照合規則に基づいて、COMPARE 関数は次の値を返します。

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

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みなしません。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、等しいとみなされません。

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]」 251 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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-expresssion-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。

構文

COMPRESS(*string-expression* **[, '***compression-algorithm-alias***'])**

パラメータ

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。

詳細については、「DECOMPRESS 関数 [文字列]」 149 ページを参照してください。

備考

通常、COMPRESS 関数は、関数に渡されたバイナリ文字列よりも短い LONG BINARY 値を返します。この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]」 149 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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' ) );
```


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を指定できます。

構文

CONNECTION_EXTENDED_PROPERTY(
{ *property-id* | *property-name* }
[, *property-specific-argument*]
)

パラメータ

property-id　接続プロパティの ID。

property-name　接続プロパティの名前。可能なプロパティ名は CharSet と NcharCharSet です。

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。

- **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。

備考

CONNECTION_EXTENDED_PROPERTY 関数は拡張の接続プロパティを返します。返される値は LONG VARCHAR であり、現在の接続に適用されます。

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。

参照

- 「接続レベルのプロパティ」『SQL Anywhere サーバ - データベース管理』
- 「CONNECTION_PROPERTY 関数 [システム]」 123 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」 144 ページ
- 「DB_PROPERTY 関数 [システム]」 148 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

integer-expression-1　ほとんどの場合は、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、「接続レベルのプロパティ」 『SQL Anywhere サーバ - 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。

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。

参照

- 「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』
- 「PROPERTY_NUMBER 関数 [システム]」 219 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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' );
```

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

column-name　競合をテストされるカラムの名前。

備考

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」 473 ページ
- 「競合の管理」 『SQL Remote』

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 トリガが起動するかどうかを決定します。RESOLVE UPDATE トリガは、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 トリガを示します。

```
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';
  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、このトリガからアクセスできる OldRemote.PKey 値がないため、このトリガは失敗します。

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;
```

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。

構文

CONVERT(*datatype*, *expression* [, *format-style*] **)**

パラメータ

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
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)

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。

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 [互換性]」『SQL Anywhere サーバ - 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]」 115 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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 ;
```

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

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。

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。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 (*x*, *y*) / **STDDEV_POP** (*x*) * **STDDEV_POP** (*y*)

x は *dependent-expression* を表し、*y* は *independent-expression* を表します。

参照

- 「集合関数」 93 ページ
- 「COVAR_POP 関数 [集合]」 131 ページ
- 「STDDEV_POP 関数 [集合]」 259 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能。

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 0.4402267564599596 を返します。

```
SELECT CORR( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) ) FROM Employees;
```

COS 関数 [数値]

数値をラジアンからコサインに変換します。

構文

COS(*numeric-expression* **)**

パラメータ

numeric-expression　角度 (ラジアン)。

備考

COS 関数は *numeric-expression* で指定される角度のコサインを返します。

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。パラメータが NULL 値の場合、結果は NULL 値になります。

参照

- 「ACOS 関数 [数値]」 103 ページ
- 「COT 関数 [数値]」 129 ページ
- 「SIN 関数 [数値]」 249 ページ
- 「TAN 関数 [数値]」 267 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

数値をラジアンからコタンジェントに変換します。

構文

COT(*numeric-expression* **)**

パラメータ

numeric-expression 角度 (ラジアン)。

備考

COT 関数は *numeric-expression* で指定される角度のコタンジェントを返します。

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。パラメータが NULL 値の場合、結果は NULL 値になります。

参照

- 「COS 関数 [数値]」 128 ページ
- 「SIN 関数 [数値]」 249 ページ
- 「TAN 関数 [数値]」 267 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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に従って、グループのロー数をカウントします。

構文 1

COUNT(
*
| *expression*
| **DISTINCT** *expression*
)

構文 2

COUNT(
{ * | *expression* }
) OVER (*window-spec* **)**

window-spec : 下の「備考」の構文 2 の説明を参照

パラメータ

***** 各グループの中のロー数を返します。

expression ローの数を返す式。

DISTINCT 式 排他ローの数を返す式。

備考

値が NULL 値のローは、カウントに含まれません。

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」 745 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 - SQL の使用法』を参照してください。

参照

- 「AVG 関数 [集合]」 107 ページ
- 「SUM 関数 [集合]」 266 ページ

標準と互換性

- **SQL/2003** コア機能。構文 2 は機能 T611 です。

例

次の文は、ユニークな各都市と、その都市の値を持つロー数を返します。

```
SELECT City , COUNT(*) 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

設定されたビットを決定するビット配列。

備考

bit-expression が NULL の場合、NULL を返します。

標準と互換性

SQL/2003　ベンダ拡張。

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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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( x * y ) - SUM( y ) * SUM( y ) / n ) / n
```

x は *dependent-expression* を表し、*y* 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、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「COVAR_SAMP 関数 [集合]」 132 ページ
- 「SUM 関数 [集合]」 266 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

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 : **下の「備考」の構文 2 の説明を参照**

パラメータ

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。

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。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、「集合関数の数学上の式」 『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「COVAR_POP 関数 [集合]」 131 ページ
- 「SUM 関数 [集合]」 266 ページ

標準と互換性

- **SQL/2003** コア SQL に含まれない SQL 基本機能 (T621)。

例

次の例は、値 74782.94600540561 を返します。

```
SELECT COVAR_SAMP( Salary, ( YEAR( NOW() ) - 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*] **)**

パラメータ

string-expression 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。

- **os_charset** 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**　データベースが使用する CHAR 文字セットのエイリアス。
- **nchar_charset**　データベースが使用する NCHAR 文字セットのエイリアス。
- **その他のサポートされた文字セット・ラベル**　SQL Anywhere がサポートする任意の文字セット・ラベルを指定できます。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 - データベース管理』を参照してください。

source-charset　元の文字列式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。

- **os_charset**　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**　データベースが使用する CHAR 文字セットのエイリアス。
- **nchar_charset**　データベースが使用する NCHAR 文字セットのエイリアス。
- **その他のサポートされた文字セット・ラベル**　SQL Anywhere がサポートする任意の文字セット・ラベルを指定できます。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 - データベース管理』を参照してください。

参照

- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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・プロシージャは、このテーブルのファイル名をパラメータとして取り出し、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;
```

ここで、mytable は、ファイル名カラムが含まれているテーブルです。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。0 ー 1 の小数値を返します。

構文

CUME_DIST( ) OVER (*window-spec* **)**

window-spec : 下の「備考」を参照

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」 745 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 - 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]」 152 ページ
- 「PERCENT_RANK 関数 [ランキング]」 214 ページ
- 「RANK 関数 [ランキング]」 222 ページ

標準と互換性

- **SQL/2003** SQL／OLAP 機能 T612

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。

構文

DATALENGTH(*expression* **)**

パラメータ

expression　通常、*expression* 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。

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/2003**　ベンダ拡張。

例

次の文は、CompanyName カラムの最も長い文字列である値 27 を返します。

```
SELECT MAX( DATALENGTH( CompanyName ) )
FROM Customers;
```

次の文は、文字列 '8sdofinsv8s7a7s7gehe4h' の長さとして、値 22 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 [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

構文

DATE(*expression* **)**

パラメータ

expression　日付フォーマットに変換される値。通常、文字列。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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]

日付にいくつかの日付の単位を加算した日付を返します。

構文

DATEADD(*date-part*, *numeric-expression*, *date-expression* **)**

date-part :
year | **quarter** | **month** | **week** | **day** | **dayofyear** | **hour** | **minute** | **second** | **millisecond**

パラメータ

date-part　日付に加算される日付の単位。日付の単位の詳細については、「日付の単位」 95 ページを参照してください。

numeric-expression　日付に加算される日付の単位の数。*numeric_expression* には任意の数値型を指定できますが、値は整数にトランケートされます。

date-expression　変更される日付。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 1995-11-02 00:00:000.000 を返します。

```
SELECT DATEADD( month, 102, '1987/05/02' );
```

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

パラメータ

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」 95 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。

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、(date2 - date1) 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。

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。

より短い時間単位のために、オーバフロー値が用意されています。

- **ミリ秒**　24 日
- **秒**　68 年
- **分**　4083 年
- **その他**　オーバフロー制限なし

これらの制限値を超えると、この関数はオーバフロー・エラーを返します。

標準と互換性

- **SQL/2003** Transact-SQL 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

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 [互換性]」 『SQL Anywhere サーバ - 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。

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/2003** ベンダ拡張。

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]

日時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*, *date-expression* **)**

パラメータ

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」 95 ページを参照してください。

date-expression 日付の単位名が返される日付。要求する *date-part* が含まれた日付を指定してください。

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。

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

日時の値の一部の値を返します。

構文

DATEPART(*date-part*, *date-expression* **)**

パラメータ

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」 95 ページを参照してください。

date-expression 値が返される日付。

備考

date-part フィールドが含まれた日付を指定してください。

曜日に対応する数値。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 [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「SET 文 [T-SQL]」 679 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 5 を返します。

```
SELECT DATEPART( month , '1987/05/02' );
```

DATETIME 関数 [日付と時刻]

式をタイムスタンプに変換します。

構文

DATETIME(*expression* **)**

パラメータ

expression 変換される式。通常は文字列です。

備考

数値を変換しようとするとエラーが返ります。

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

1 ー 31 の整数を返します。

構文

DAY(*date-expression* **)**

パラメータ

date-expression　日付。

備考

日付の月日に対応する整数 1 ー 31。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

date-expression　日付。

備考

返される曜日名は英語で、Sunday、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday です。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 Saturday を返します。

```
SELECT DAYNAME ( '1987/05/02' );
```

DAYS 関数 [日付と時刻]

日付を評価する関数。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：整数

DAYS([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

DAYS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する日数。*integer-expression* が負の場合、タイムスタンプから適切な日数が引かれます。整数式を指定する場合は、*datetime-expression* を日付またはタイムスタンプとして明示的にキャストしてください。

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]」 115 ページを参照してください。

備考

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの日数を返します。

  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。日付アルゴリズムで使用される日付です。

- 2 つの日付を指定すると、2 つの日付の間の日数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- この関数で 1 つの日付と整数を指定すると、指定した日付に、指定した整数の日数が加算されます。代わりに、DATEADD 関数を使用します。

この関数では、時間、分、秒が無視されます。

参照

- 「DATEDIFF 関数 [日付と時刻]」 138 ページ
- 「DATEADD 関数 [日付と時刻]」 137 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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。これは、2 番目の日付が最初の日付より 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、タイムスタンプ 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

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のリストについては、「データベース・レベルのプロパティ」『SQL Anywhere サーバ - 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。
  - **AccentSensitivity**　AccentSensitivity 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', 'AccentSensitivity');
    ```

    戻り値は、Ignore、Respect、French のいずれかです。これらの値の説明については、「照合の適合化オプション」 382 ページ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。これらの値の説明については、「照合の適合化オプション」 382 ページ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。これらの値の説明については、「照合の適合化オプション」 382 ページ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」 382 ページを参照してください。
- **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」 382 ページ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。
- **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。

備考

LONG VARCHAR 型の値を返します。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。

DB_EXTENDED_PROPERTY 関数は、*property-specific-argument* 文字列パラメータ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。

テーブル名やプロシージャ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]」 147 ページ
- 「DB_NAME 関数 [システム]」 147 ページ
- 「データベース・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』
- 「CONNECTION_PROPERTY 関数 [システム]」 123 ページ
- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」 122 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

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' );
```

文 SELECT DB_EXTENDED_PROPERTY ('Collation', 'Properties'); 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。

```
'CaseSensitivity=Ignore'
```

文 SELECT DB_EXTENDED_PROPERTY('NCharCollation', 'Specification'); は、データベースの NCHAR 照合の完全照合指定を返します。

```
'UCA(CaseSensitivity=Ignore;AccentSensitivity=Ignore;PunctuationSensitivity=Primary)'
```

DB_ID 関数 [システム]

データベース ID 番号を返します。

構文

DB_ID([*database-name*] **)**

パラメータ

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。

参照

- 「global_database_id オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

サーバ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

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。

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。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

サーバ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。

構文

DB_PROPERTY(
{ *property-id* | *property-name* }
[, *database-id* | *database-name*]
)

パラメータ

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。

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。

参照

- 「DB_ID 関数 [システム]」 147 ページ
- 「DB_NAME 関数 [システム]」 147 ページ
- 「データベース・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

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

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]」 121 ページを参照してください。

備考

DECOMPRESS 関数は LONG BINARY 値を返します。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・サーバが自動的に処理します。「カラムを圧縮するかどうかの選択」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

参照

- 「COMPRESS 関数 [文字列]」 121 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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;
```

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*]
)

パラメータ

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、パラメータの大文字と小文字は区別されます。

> **警告**
> キーは保護してください。キーのコピーは安全な場所に保管してください。キーを紛失すると、暗号化データにまったく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。

algorithm　このオプションのパラメータでは、*string-expression* の復号化に使用されるアルゴリズムを指定します。*string-expression* は、暗号化で使用されたのと同じアルゴリズムを使用して復号化します。Rijndael は、SQL Anywhere の強力な暗号化を実装するため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 をサポートしているプラットフォームでは、AES_FIPS タイプを使用して、別個の FIPS 認定 AES アルゴリズムを強力な暗号化として指定することもできます。-fips オプションを指定してデータベース・サーバを起動する場合、AES または AES_FIPS の強力な暗号化方式で暗号化されたデータベースを実行できますが、単純暗号化方式で暗号化されたデータベースは実行できません。-fips が指定されている場合、暗号化されていないデータベースもサーバで開始できます。

備考

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。

参照

- 「ENCRYPT 関数 [文字列]」 154 ページ
- 「データベースの一部の暗号化」 『SQL Anywhere サーバ - データベース管理』
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、user_info テーブルにあるユーザ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

numeric-expression　角度 (ラジアン)。

備考

DEGREES 関数は *numeric-expression* で指定される角度を返します。

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。パラメータが NULL 値の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 29.79380534680281 を返します。

```
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。同順の値の場合、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 : 下の「備考」を参照

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」 745 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 - 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]」 135 ページ
- 「PERCENT_RANK 関数 [ランキング]」 214 ページ
- 「RANK 関数 [ランキング]」 222 ページ

標準と互換性

- **SQL/2003** SQL／OLAP 機能 T612

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。

Surname	Salary	State	SalaryRank
Shishov	72995.000	UT	1
Wang	68400.000	UT	2
Cobb	62000.000	UT	3
Morris	61300.000	UT	4

Surname	Salary	State	SalaryRank
Davidson	57090.000	NY	5
Martel	55700.000	NY	6
Blaikie	54900.000	NY	7
Diaz	54900.000	UT	7
Driscoll	48023.000	UT	8
Hildebrand	45829.000	UT	9
Whitney	45700.000	NY	10
Guevara	42998.000	NY	11
Soo	39075.000	NY	12
Goggin	37900.000	UT	13
Wetherby	35745.000	NY	14
Ahmed	34992.000	NY	15
Rebeiro	34576.000	UT	16
Bigelow	31200.000	UT	17
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

string-expression-1　最初の SOUNDEX 引数。

string-expression-2　2 番目の SOUNDEX 引数。

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 ー 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]」 255 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 3 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 ー 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

date-expression 評価する日付。

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/2003** ベンダ拡張。

例

次の文は、値 5 を返します。

```
SELECT DOW( '1998-07-09' );
```

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*]
)

パラメータ

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
キーは保護してください。キーのコピーは、安全な場所に保管してください。キーを紛失すると、暗号化データにまったく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。

algorithm　このオプションのパラメータでは、*string-expression* の暗号化に使用されるアルゴリズムを指定します。*string-expression* は、暗号化で使用されたのと同じアルゴリズムを使用して復号化します。Rijndael は、SQL Anywhere の強力な暗号化を実装するため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 をサポートしているプラットフォームでは、AES_FIPS アルゴリズムを使用して、別個の FIPS 認定 AES アルゴリズムを強力な暗号化として指定することもできます。

備考

この関数は、入力 *string-expression* より最大 31 バイト長い LONG BINARY 値を返します。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]」 150 ページ
- 「データベースの一部の暗号化」 『SQL Anywhere サーバ - データベース管理』
- 「-fips サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能。

例

次のトリガは、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のパスワードが含まれ、トリガは、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

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・メッセージを含む文字列。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 (%1) で指定されます。

参照

- 「エラー・メッセージ (SQLSTATE 順)」『SQL Anywhere 10 - エラー・メッセージ』
- 「エラー・メッセージ (SQL Anywhere の SQLCODE 順)」『SQL Anywhere 10 - エラー・メッセージ』

標準と互換性

- **SQL/2003**　ベンダ拡張。

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に基づいて、クエリ・オプティマイザの選択性推定を提供します。

構文

ESTIMATE(*column-name* [, *value* [, *relation-string*]] **)**

パラメータ

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。このパラメータに使用できる値は、=、>、<、>=、<=、<>、!=、!<、!> です。デフォルトは = です。

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

- 「INDEX_ESTIMATE 関数 [その他]」 185 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」 157 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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;
```

ESTIMATE_SOURCE 関数 [その他]

クエリ・オプティマイザ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

column-name 調査されるカラムの名前。

値 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。このパラメータに使用できる値は、=、>、<、>=、<=、<>、!=、!<、!> です。デフォルトは = です。

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。

戻り値

選択性推定のソースは、次のいずれかです。

- **Statistics**　は、値が指定され、カラムの値の平均による選択性を推定する統計情報が格納されている場合に使用されます。統計情報を使用できるのは、指定した値の選択性が統計情報に格納されるだけ重大な数値である場合だけです。現在、値が重大であるとみなされるのは、少なくともローの 1% で値が出現する場合です。
- **Column**　は、値の選択性がローの 1% 未満で出現すること以外は、Statistics とほぼ同じです。この場合、使用される選択性は、統計情報に格納された、ローの 1% 未満で出現するすべての値の平均値です。
- **Guess**　は、使用できる適切なインデックスがなく、カラムの統計情報がまったく収集されていない場合に返されます。この場合は、組み込み規則が使用されます。
- **Column-column**　は、使用される推定がジョインの選択性の場合に返されます。この場合、推定は、ジョインの結果セットのロー数を 2 つのテーブルの直積のロー数で割った値です。
- **Index**　は、選択性の推定に使用できる統計情報はないが、選択性推定のためにインデックスを調査できる場合に使用されます。
- **User**　は、ユーザ指定の推定が存在し、user_estimates データベース・オプションが Disabled に設定されていない場合に返されます。

  詳細については、「user_estimates オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **Computed**　は、他の情報に基づいて、オプティマイザが統計情報を計算する場合に返されます。たとえば、SQL Anywhere は複数のカラムの統計情報を管理しません。したがって、x=5 と y=10 などの複数のカラムの等式に関する推定が必要なとき、カラム x とカラム y の統計情報が存在する場合は、オプティマイザが各カラムの推定した選択性を乗算して推定値を生成します。
- **Always**　は、テストが明らかに真の場合に使用されます。たとえば、値が 1=1 の場合です。
- **Combined**　は、オプティマイザが前述の複数のソースを使用し、それらを組み合わせる場合に使用されます。
- **Bounded**　は、他のいずれかのソースに制限を加えることができます。Bounded は、SQL Anywhere が推定に上限か下限またはその両方を設定したことを示します。オプティマイザは、推定値が論理範囲を超えないようにするためにこれらの制限を設定します。たとえば、推定値が 100% を超えたり、選択性が 1 ローより小さくならないことを保証します。

参照

- 「ESTIMATE 関数 [その他]」 157 ページ
- 「INDEX_ESTIMATE 関数 [その他]」 185 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 Index を返します。これは、クエリ・オプティマイザがインデックスを調査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;
```

EVENT_CONDITION 関数 [システム]

イベント・ハンドラがトリガされる条件を指定します。

構文

EVENT_CONDITION(*condition-name* **)**

パラメータ

condition-name　イベントをトリガする条件。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	

条件名	単位	イベント	コメント
TempFreeSpace	MB	TempDiskSpace	
TempSize	MB	GrowTemp	

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。

参照

- 「CREATE EVENT 文」 397 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

この関数を使用して、EVENT_CONDITION に指定可能なパラメータをリストできます。

構文

EVENT_CONDITION_NAME(*integer* **)**

パラメータ

integer 0 以上の整数。

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」 397 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

EVENT_PARAMETER 関数 [システム]

イベント・ハンドラ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'**
| **'TableName'**
| **'User'**
| *condition-name*

パラメータ

context-name　あらかじめ設定されている文字列の 1 つ。文字列は大文字と小文字を区別しません。次の情報を持っています。

- **AppInfo**　イベントをトリガさせた接続の AppInfo 接続プロパティの値。次の文を使用すると、イベントのコンテキスト外でもプロパティの値を参照できます。

  ```
  SELECT connection_property( 'AppInfo' );
  ```

  このパラメータは、Connect、Disconnect、ConnectFailed、BackupEnd、RAISERROR の各イベントに対して有効です。AppInfo 文字列には、Embedded SQL、ODBC、OLE DB、ADO.NET、iAnywhere JDBC ドライバの各接続に対するクライアント接続マシン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。
  - **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。
  - **drop connection**　DROP CONNECTION 文が実行されました。
  - **liveness**　-tl サーバ・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。
  - **inactive**　-ti サーバ・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。
  - **connect failed**　接続の試行に失敗しました。
- **EventName**　トリガされたイベント名。
- **Executions**　イベント・ハンドラの実行回数。

- **MirrorServerName** データベース・ミラーリング・システムのプライマリ・サーバとの接続を失ったミラー・サーバまたは監視サーバの名前。
- **NumActive** イベント・ハンドラのアクティブ・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。
- **TableName** RemainingValues で使用するテーブル名。
- **User** イベントをトリガさせたユーザのユーザ 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。

Context-name 値	有効なシステム・イベント・タイプ
AppInfo	BackupEnd、"Connect"、ConnectFailed、"Disconnect"、"RAISERROR"、ユーザ・イベント
ConnectionID	BackupEnd、"Connect"、"Disconnect"、Global Autoincrement、"RAISERROR"、ユーザ・イベント
DisconnectReason	"Disconnect"
EventName	すべて
Executions	すべて
NumActive	すべて
TableName	GlobalAutoincrement
User	BackupEnd、"Connect"、ConnectFailed、"Disconnect"、Global Autoincrement、"RAISERROR"、ユーザ・イベント

参照

- 「EVENT_CONDITION 関数 [システム]」 159 ページ
- 「CREATE EVENT 文」 397 ページ
- 「TRIGGER EVENT 文」 717 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例は、イベントに文字列パラメータを渡す方法を示します。イベントは、トリガされた時刻をサーバ・コンソールに表示します。

```
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]

指数関数 (e を底とする数のべき乗) を返します。

構文

EXP(*numeric-expression* **)**

パラメータ

numeric-expression　指数。

備考

EXP 関数は、*numeric-expression* で指定された値の指数値を返します。

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。パラメータが NULL 値の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 3269017.3724721107 を返します。

```
SELECT EXP( 15 );
```

EXPERIENCE_ESTIMATE 関数 [その他]

この関数は常に頻度テーブルを調べることを除き、ESTIMATE 関数と同じです。

構文

EXPERIENCE_ESTIMATE(
column-name
[, *value*
[, *relation-string*]]
)

パラメータ

column-name　調査されるカラムの名前。

値　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。このパラメータに使用できる値は、=、>、<、>=、<=、<>、!=、!<、!> です。デフォルトは = です。

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]」 157 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 文のプランの最適化方法を返します。

構文

EXPLANATION(
string-expression
[*cursor-type*],
[*update-status*]
)

パラメータ

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。

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。これらの値は、指定されたカーソルをオプティマイザ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。

?値	説明
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。

備考

最適化結果は文字列として返されます。

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。

Interactive SQL では、SQL 文のプランを [結果] ウィンドウ枠の [プラン] タブに表示できます。

参照

- 「Ultra Light のクエリ・アクセス・プラン」『Ultra Light - データベース管理とリファレンス』
- 「実行プランの解釈」『SQL Anywhere サーバ - SQL の使用法』
- 「PLAN 関数 [その他]」 216 ページ
- 「GRAPHICAL_PLAN 関数 [その他]」 171 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、SELECT 文を文字列パラメータとして渡し、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

string-expression　SELECT 文。式のデータ型が問い合わせされる場合、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。

参照

- 「SQL データ型」 47 ページ
- 「sa_describe_query システム・プロシージャ」 891 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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(*expression* [**IGNORE NULLS**] **)**
OVER (*window-spec* **)**

window-spec : 下の「備考」を参照

パラメータ

expression　評価する式。たとえば、カラム名です。

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。IGNORE NULLS を指定しないと、(NULL であるかどうかにかかわらず) 先頭の値が返されます。

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。

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」 745 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 - 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 - SQL の使用法』

- 「LAST_VALUE 関数 [集合]」 189 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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;
```

以下に示す結果セットでは、従業員 1658 は部署 500 の最初のローに示されているため、この部署内で最近採用された従業員であることがわかります。したがって、パーセンテージは 100% に設定されています。部署 500 の残りの従業員のパーセンテージは、従業員 1658 に対して相対的に比較されて算出されます。たとえば、従業員 1570 は、従業員 1658 の給料の約 139% に該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。

DepartmentID	EmployeeID	percentage
500	1658	100
500	1615	110.4284624
500	1570	138.8427097
500	1013	109.5851905
500	921	167.4497049
500	868	113.2393688
500	750	137.7344095
500	703	222.8679276
500	191	119.6642975
400	1751	100
400	1740	99.705647
400	1684	130.969936
400	1643	83.9734797

DepartmentID	EmployeeID	percentage
400	1607	175.1828989
400	1576	197.0164609
...	...	...

FLOOR 関数 [数値]

数値の切り捨て値 (値以下の最大整数値) を返します。

構文

FLOOR(*numeric-expression* **)**

パラメータ

numeric-expression　値を指定します。通常は FLOAT です。

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。

参照

- 「CEILING 関数 [数値]」 115 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、123 の Floor 値を返します。

```
SELECT FLOOR (123);
```

次の文は、123 の Floor 値を返します。

```
SELECT FLOOR (123.45);
```

次の文は、-124 の Floor 値を返します。

```
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

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。

備考

配列の先頭ビットの位置を 1 とします。

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」 13 ページ
- 「SET_BIT 関数 [ビット配列]」 245 ページ
- 「SET_BITS 関数 [集合]」 246 ページ
- 「sa_get_bits システム・プロシージャ」 900 ページ

標準と互換性

SQL/2003　ベンダ拡張。

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]

オートインクリメント・カラムに値を割り当てます。オートインクリメント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

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　ID に割り当てる値の開始値。デフォルトは 1 です。

備考

最も効率的な ID の生成方法はオートインクリメントまたはグローバル・オートインクリメント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、テーブルにオートインクリメント・カラムが定義されていることを前提としています。テーブルのオートインクリメント・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、テーブルにオートインクリメント・カラムが存在しない場合は NULL を返します。複数のオートインクリメント・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。

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 - SQL の使用法』を参照してください。

参照

- 「CREATE TABLE 文」 460 ページ
- 「ALTER TABLE 文」 336 ページ
- 「NUMBER 関数 [その他]」 212 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、テーブルのオートインクリメント・カラムに使用できる次の数値を返し、その数値と以下に示す 9 個の値を保存します。

```
SELECT GET_IDENTITY( 'GROUPO.T2', 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( )

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

- 「NOW 関数 [日付と時刻]」 211 ページ
- 「CURRENT TIMESTAMP 特別値」 31 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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	オプティマイザの推定のみ (デフォルト)
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。これらの値は、指定されたカーソルをオプティマイザ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。

値	説明
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。

参照

- 「PLAN 関数 [その他]」 216 ページ
- 「EXPLANATION 関数 [その他]」 164 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の Interactive SQL の例は、SELECT 文を文字列パラメータとして渡し、クエリを実行するためのプランを返します。プランは *plan.xml* ファイルに保存されます。

```
SELECT GRAPHICAL_PLAN(
  'SELECT * FROM Departments WHERE DepartmentID > 100' );
OUTPUT TO plan.xml FORMAT FIXED;
```

次の文は、'SELECT * FROM Department 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' );
```

Interactive SQL では、SQL 文のプランを [結果] ウィンドウ枠の [プラン] タブに表示できます。

GREATER 関数 [その他]

2 つのパラメータ値のうち、より大きい値を返します。

構文

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

パラメータ

expression-1 比較される最初のパラメータ値。

expression-2 比較される 2 番目のパラメータ値。

備考

パラメータが等しい場合は、最初のパラメータを返します。

参照

- 「LESSER 関数 [その他]」 193 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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* **)**

パラメータ

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 の使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「CUBE の使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「GROUP BY GROUPING SETS」 『SQL Anywhere サーバ - SQL の使用法』
- 「SELECT 文」 669 ページ
- 「GROUPING 関数を使用したプレースホルダの NULL の検出」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T611)。

例

この関数の使用例については、「GROUPING 関数を使用したプレースホルダの NULL の検出」 『SQL Anywhere サーバ - 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

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 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。

注意
FIPS アルゴリズムは、Certicom の FIPS 140-2 基準を満たしたソフトウェアを使用するシステムでのみ使用されます。データベース・サーバを起動するときに FIPS アルゴリズムを使用する場合には、-fips オプションを指定する必要があります。

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。

サーバを -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は次のように異なる可能性があります。

- 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。

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。

- 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(40) を返します
- SHA256_FIPS は VARCHAR(40) を返します

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」 99 ページ
- 「-fips サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、ユーザ ID やパスワードなど、アプリケーションのユーザ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。

```
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。

構文

HEXTOINT(*hexadecimal-string* **)**

パラメータ

hexadecimal-string　整数に変換される文字列。

備考

HEXTOINT 関数は、数値、大文字または小文字の A ー 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。

HEXTOINT 関数は、プラットフォームに依存しない SQL INTEGER 相当の 16 進文字列を返します。右から 8 桁目が数値 8 ー 9 か大文字または小文字の A ー F で、その前の桁がすべて大文字または小文字の F の場合、16 進値は負の整数値になります。次の HEXTOINT は無効な使用例です。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' );
```

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]」 186 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 420 を返します。

```
SELECT HEXTOINT( '1A4' );
```

HOUR 関数 [日付と時刻]

日時の時間コンポーネントを返します。

構文

HOUR(*datetime-expression* **)**

パラメータ

datetime-expression　日時。

備考

返される値は日時の時間に対応する 0 ー 23 の数値です。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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]

時間を評価する関数。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：整数

HOURS ([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

HOURS (*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する時間数。*integer-expression* が負の場合、日時から適切な時間数が引かれます。整数式を指定する場合は、*datetime-expression* を DATETIME データ型として明示的にキャストしてください。

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]」 115 ページを参照してください。

備考

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの時間数を返します。

  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。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つの時刻の間の時間数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定した日付に、指定した整数の時間数が加算されます。代わりに、DATEADD 関数を使用します。

参照

- 「DATEDIFF 関数 [日付と時刻]」 138 ページ
- 「DATEADD 関数 [日付と時刻]」 137 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 4 を返します。これは、2 番目のタイムスタンプが、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

string　HTML ドキュメントで使用される任意のリテラル文字列。

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
&quot;	"
&#39;	'
&amp;	&
&lt;	<
&gt;	>
&#x*hexadecimal-number*;	ユニコードのコードポイント。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*;	ユニコードのコードポイント。10 進数で指定します。たとえば、™ は商標記号を返します。

指定したユニコード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/ を参照してください。

参照

- 「HTML_ENCODE 関数 [その他]」 178 ページ
- 「HTTP_DECODE 関数 [HTTP]」 179 ページ
- 「HTTP_ENCODE 関数 [HTTP]」 180 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

string　HTML ドキュメントで使用される任意の文字列。

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
"	&quot;
'	&#39;
&	&amp;
<	&lt;
>	&gt;
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]」 178 ページ
- 「HTTP_ENCODE 関数 [HTTP]」 180 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

HTTP_DECODE 関数 [HTTP]

HTTP で使用する文字列内の特殊文字を復号化します。

構文

HTTP_DECODE(*string* **)**

パラメータ

string　HTTP 要求で使用される任意の文字列。

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 [HTTP]」 180 ページ
- 「HTML_DECODE 関数 [その他]」 178 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

HTTP_ENCODE 関数 [HTTP]

HTTP で使用する文字列内の特殊文字をエンコードします。

構文

HTTP_ENCODE(*string* **)**

パラメータ

string HTTP 要求で使用される任意の文字列。

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 1F 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

文字	置換
|	%7C
}	%7D
0x1f 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 [HTTP]」 179 ページ
- 「HTML_ENCODE 関数 [その他]」 178 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

HTTP_HEADER 関数 [HTTP]

HTTP ヘッダの値を取得します。

構文

HTTP_HEADER(*header-field-name* **)**

パラメータ

header-field-name HTTP ヘッダ・フィールドの名前。

備考

この関数は、指定された HTTP ヘッダ・フィールドの値を返します。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。

指定した *header-field-name* のヘッダが存在しない場合、戻り値は NULL です。関数が Web サービスから呼び出されていない場合にも戻り値は NULL になります。

HTTP Web サービスの要求を処理するときに、次のような関連するヘッダがあります。このようなヘッダの詳細については、http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.org/Protocols/rfc2616/rfc2616-sec14.html を参照してください。

- **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。

HTTP Web サービス要求を処理するときは、常に 3 つの特殊なヘッダが定義されます。

- **@HttpMethod**　処理されている要求の種類を返します。可能な値には HEAD、GET、POST があります。
- **@HttpURI**　HTTP 要求で指定された、要求の完全な URI。
- **@HttpVersion**　要求の HTTP バージョン (たとえば、1.0 または 1.1)。

これらの特殊なヘッダを使用して、クライアント要求の最初の行 (要求行と呼ばれます) にアクセスできます。

参照

- 「HTTP_VARIABLE 関数 [HTTP]」 182 ページ
- 「NEXT_HTTP_HEADER 関数 [HTTP]」 209 ページ
- 「NEXT_HTTP_VARIABLE 関数 [HTTP]」 210 ページ
- 「HTTP ヘッダの使用」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例では、Cookie のヘッダ値を取得します。

```
SET cookie_value = HTTP_HEADER( 'Cookie' );
```

次の例は、最初の HTTP ヘッダ値を返します。

```
DECLARE header_name LONG VARCHAR;
DECLARE header_value LONG VARCHAR;
SET header_name = NEXT_HTTP_HEADER( NULL );
SET header_value = HTTP_HEADER( header_name );
```

HTTP_VARIABLE 関数 [HTTP]

HTTP 変数の値を取得します。

構文

HTTP_VARIABLE(*var-name* [[, *instance*] , *http-header-field*] **)**

パラメータ

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。

http-header-field　マルチパートの要求の中で、*var-name* で指定されたフィールドに関連づけられたヘッダ・フィールド名。

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。

指定した *var-name* のヘッダが存在しない場合、戻り値は NULL です。

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 サーバは個々の変数の HTTP ヘッダを受信します。*http-header-field* パラメータを指定すると、特定の変数の POST 要求から関連する multipart/form-data ヘッダ値を HTTP_VARIABLE 関数から返されます。

クライアント (ブラウザなど) 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、*http-header-field* に @BINARY を指定すると、文字セットが変換されていない変数の入力値が返されます。@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。

Web サービスから呼び出されていない場合、この関数は NULL を返します。

参照

- 「HTTP_HEADER 関数 [HTTP]」 181 ページ
- 「NEXT_HTTP_HEADER 関数 [HTTP]」 209 ページ
- 「NEXT_HTTP_VARIABLE 関数 [HTTP]」 210 ページ
- 「変数の使用」『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、イメージ変数の Content-Disposition ヘッダと Content-Type ヘッダを要求します。

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
```

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

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。

備考

IDENTITY 関数の使用方法については、「NUMBER 関数 [その他]」 212 ページを参照してください。

参照

- 「NUMBER 関数 [その他]」 212 ページ

標準と互換性

- **SQL/2003** Transact-SQL 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 NULL 以外の式を返します。

構文

IFNULL(*expression-1*, *expression-2* [, *expression-3*] **)**

パラメータ

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。

備考

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。

標準と互換性

- **SQL/2003** Transact-SQL 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]

この関数は常にインデックスのみを調べることを除き、ESTIMATE 関数と同じです。

構文

INDEX_ESTIMATE(*column-name*, *number*
[, *relation-string*]
)

パラメータ

column-name　推定で使用されるカラムの名前。

number　*number* を指定すると、クエリ・オプティマイザが使用する推定のパーセンテージを REAL として返します。

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。このパラメータに使用できる値は、'='、'>'、'<'、'>='、'<='、'<>'、'!='、'!<'、'!>' です。デフォルトは '=' です。

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]」 157 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」 157 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 89.4736862183 を返します。

```
SELECT FIRST 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

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。

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]」 263 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

integer-expression 16 進に変換される整数。

参照

- 「HEXTOINT 関数 [データ型変換]」 175 ページ

標準と互換性

- **SQL/2003** Transact-SQL 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

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。

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。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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;
```

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

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となっている必要があります。

備考

この関数の戻り値は、指定した式によって異なります。データベース・サーバ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 (NULL でない最初の式)。データベース・サーバは、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]」 118 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。

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る場合は 1、そうでない場合は 0 を返します。文字列に空白だけが含まれるか、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/2003**　ベンダ拡張。

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(*expression* [**IGNORE NULLS**] **)**
OVER (*window-spec* **)**

window-spec : 下の「備考」を参照

パラメータ

expression　評価する式。たとえば、カラム名です。

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。IGNORE NULLS を指定しないと、(NULL であるかどうかにかかわらず) 最後の値が返されます。

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」 745 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 - 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 - SQL の使用法』
- 「FIRST_VALUE 関数 [集合]」 166 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 );
```

以下に示す結果セットでは、部署 500 で最高額の給料を受け取っている従業員は Jose Martinez であることを示しています。また、部署 400 で最高額の給料を受け取っている従業員は Scott Evans であることを示しています。

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'
...	...	...	...

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。この関数は LOWER 関数と同じです。

構文

LCASE(*string-expression* **)**

パラメータ

string-expression　小文字に変換される文字列。

参照

- 「LOWER 関数 [文字列]」 198 ページ
- 「UCASE 関数 [文字列]」 273 ページ
- 「UPPER 関数 [文字列]」 275 ページ
- 「文字列関数」 99 ページ

備考

LCASE 関数は LOWER 関数と同じです。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

string-expression　文字列。

integer-expression　返す文字数。

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、戻り値は文字長のセマンティックの観点から記述されます。

参照

- 「RIGHT 関数 [文字列]」 239 ページ

- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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(*string-expression* **)**

パラメータ

string-expression 文字列。

備考

この関数を使用すると、文字列の長さをがわかります。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。

> **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]」 113 ページ
- 「国際言語と文字セット」 『SQL Anywhere サーバ - データベース管理』
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

expression-1　比較される最初のパラメータ値。

expression-2　比較される 2 番目のパラメータ値。

備考

パラメータが等しい場合は、最初の値を返します。

参照

- 「GREATER 関数 [その他]」 172 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 5 を返します。

```
SELECT LESSER( 10, 5 ) FROM dummy;
```

LIST 関数 [集合]

カンマで区切られた値のリストを返します。

構文

LIST(
{ *string-expression* | **DISTINCT** *string-expression* }
[**,** *delimiter-string*]
[**ORDER BY** *order-by-expression*] **)**

パラメータ

string-expression　文字列。通常はカラム名です。各ローに対して、カンマで区切られたリストに式の値が追加されます。

DISTINCT 文字列式　クエリで使用するカラム名などの式。そのカラムのユニークな値が、カンマで区切られたリストに 1 つずつ追加されます。

delimiter-string　リスト項目のデリミタ文字列。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。

同じクエリ・ブロック内の複数の LIST 関数が、異なる *order-by-expression* の引数を使用することはできません。

備考

NULL 値は、リストに追加されません。LIST (X) は、グループの各ローの、NULL 以外のすべての X の値を (デリミタ付きで) 連結させて返します。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。

DISTINCT と ORDER BY の両方を指定する場合、DISTINCT 式と ORDER BY 式は同じにしてください。

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/2003**　ベンダ拡張。

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;
```

ソートされた ID '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;…

次の文と前の文を区別してください。次の文は、(Surname, ';') の複合ソート・キーによってソートされた従業員 ID のカンマで区切られたリストを返します。

```
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

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。

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」 99 ページ
- 「CHARINDEX 関数 [文字列]」 117 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

numeric-expression　数値。

参照

- 「LOG10 関数 [数値]」 197 ページ

備考

引数は、組み込みの数値データ型の値を返す式です。

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。パラメータが NULL 値の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

numeric-expression　数値。

備考

引数は、組み込みの数値データ型の値を返す式です。

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。パラメータが NULL 値の場合、結果は NULL 値になります。

参照

- 「LOG 関数 [数値]」 196 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

string-expression　変換される文字列。

備考

LCASE 関数は LOWER 関数と同じです。

参照

- 「LCASE 関数 [文字列]」 190 ページ
- 「UCASE 関数 [文字列]」 273 ページ
- 「UPPER 関数 [文字列]」 275 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　コア機能。

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

string-expression　削除される文字列。

備考

結果の実際の長さは、式の長さから、削除される文字数を引いた値です。すべての文字を削除すると、結果は空の文字列です。

パラメータに NULL を指定できる場合、結果は NULL になります。

パラメータが 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]」 243 ページ
- 「TRIM 関数 [文字列]」 272 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

  SQL/2003 規格で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数から指定されます。

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(*expression* | **DISTINCT** *expression* **)**

構文 2

MAX(*expression* **) OVER (** *window-spec* **)**

window-spec : **下の「備考」の構文 2 の説明を参照**

パラメータ

expression 最大値が計算される式。通常はカラム名です。

DISTINCT 式 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。

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」 745 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 - 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。

参照

- 「MIN 関数 [集合]」 200 ページ

標準と互換性

- **SQL/2003** コア機能。構文 2 は機能 T611 です。

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;
```

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(*expression* | **DISTINCT** *expression* **)**

構文 2

MIN(*expression* **) OVER (** *window-spec* **)**

window-spec : **下の「備考」の構文 2 の説明を参照**

パラメータ

expression 最小値が計算される式。通常はカラム名です。

DISTINCT 式 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。

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」 745 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 - 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。

参照

- 「MAX 関数 [集合]」 199 ページ

標準と互換性

- **SQL/2003**　コア機能。構文 2 は機能 T611 です。

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]

日時値の分コンポーネントを返します。

構文

MINUTE(*datetime-expression* **)**

パラメータ

datetime-expression　日時の値。

備考

返される値は日時の分に対応する 0 ー 59 の数値です。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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]

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの分数を返します。

  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。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つの時刻の間の分数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定した日付に、指定した整数の分数が加算されます。代わりに、DATEADD 関数を使用します。

構文 1：整数

MINUTES([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

MINUTES(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する分数。*integer-expression* が負の場合、日時の値から適切な分数が引かれます。整数式を指定する場合は、*datetime-expression* を DATETIME データ型として明示的にキャストしてください。

備考

この関数は整数を返すので、構文 1 で 4083-03-23 02:08:00 以上のタイムスタンプを使用すると、オーバフローが起こる場合があります。

参照

- 「CAST 関数 [データ型変換]」 115 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 240 を返します。これは、2 番目のタイムスタンプが、最初のタイムスタンプの 240 秒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、タイムスタンプ 1999-05-12 21:10:07.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5);

SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```

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

dividend　被除数 (分数の分子)。

divisor　除数 (分数の分母)。

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]」 234 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能。

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

date-expression　日時の値。

備考

返される値は日時の月に対応する 1 ー 12 の数値です。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

date-expression　日時の値。

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。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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]

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02 からの月数を返します。

  注意
  0000-02 は実際の日付を指すための値ではありません。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つの日付の間の月数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定した日付に、指定した整数の分数が加算されます。代わりに、DATEADD 関数を使用します。

構文 1：整数

MONTHS([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

MONTHS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する月数。*integer-expression* が負の場合、日時の値から適切な月数が引かれます。*integer-expression* を指定する場合は、*datetime-expression* を datetime データ型として明示的にキャストしてください。

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]」 115 ページを参照してください。

備考

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。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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、タイムスタンプ 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]

ユニコードのコードポイントがパラメータに指定された 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

integer　対応するユニコード・コードポイントに変換される数値。

参照

- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」 122 ページ
- 「TO_NCHAR 関数 [文字列]」 269 ページ

- 「TO_CHAR 関数 [文字列]」 268 ページ
- 「UNICODE 関数 [文字列]」 274 ページ
- 「UNISTR 関数 [文字列]」 274 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例は、アラビア文字の ALEF (ユニコード・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

NEWID 関数に関連付けられているパラメータはありません。

備考

NEWID 関数は、ユニークな識別子の値を生成します。この関数は、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。あるコンピュータで生成される値は、他のコンピュータで生成される値と一致しないようになっています。したがって、同期環境やレプリケーション環境で、これらをキーとして使用することもできます。

他の RDBMS と互換性を保つために、デフォルトの 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 [データベース]」 『SQL Anywhere サーバ - 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る可能性があります。クエリ・オプティマイザは、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 - 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 - SQL の使用法』
- 「STRTOUUID 関数 [文字列]」 263 ページ
- 「UUIDTOSTR 関数 [文字列]」 276 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 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*] **)**

パラメータ

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。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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。

```
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({ NULL | *database-id* **})**

パラメータ

database-id　データベースの ID 番号を指定する整数。

備考

NEXT_DATABASE 関数を使用して、データベース・サーバ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、サーバへの接続が行われた順序を知ることができます。サーバに接続する最初のデータベースには値 0 が割り当てられ、サーバへのその後の接続では、データベース ID が値 1 で増分されます。

標準と互換性

- **SQL/2003** Transact-SQL 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、NULL を返します。これは、サーバ上にこれ以上のデータベースがないことを示します。

```
SELECT NEXT_DATABASE( 0 );
```

NEXT_HTTP_HEADER 関数 [HTTP]

次の HTTP ヘッダの名前を取得します。

構文

NEXT_HTTP_HEADER(*header-name* **)**

パラメータ

header-name 前のヘッダの名前。header-name が NULL の場合、この関数は最初の HTTP ヘッダの名前を返します。

備考

この関数は、要求に含まれる HTTP ヘッダに対して反復され、次の HTTP ヘッダ名を返します。NULL を指定して呼び出すと、最初のヘッダの名前が返されます。後続のヘッダは、関数に前のヘッダの名前を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・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 [HTTP]」 181 ページ
- 「HTTP_VARIABLE 関数 [HTTP]」 182 ページ
- 「NEXT_HTTP_VARIABLE 関数 [HTTP]」 210 ページ
- 「HTTP ヘッダの使用」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例は、最初の HTTP ヘッダ名を取得します。

```
DECLARE header_name LONG VARCHAR;
SET header_name  = NULL;
SET header_name = NEXT_HTTP_HEADER( header_name );
```


NEXT_HTTP_VARIABLE 関数 [HTTP]

次の HTTP 変数の名前を取得します。

構文

NEXT_HTTP_VARIABLE(*var-name* **)**

パラメータ

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。

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。この関数を最後のヘッダ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_HEADER 関数 [HTTP]」 181 ページ
- 「HTTP_VARIABLE 関数 [HTTP]」 182 ページ
- 「NEXT_HTTP_HEADER 関数 [HTTP]」 209 ページ
- 「変数の使用」『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、最初の HTTP 変数名を取得します。

```
DECLARE variable_name LONG VARCHAR;
SET variable_name = NULL;
SET variable_name = NEXT_HTTP_VARIABLE( variable_name );
```

NEXT_SOAP_HEADER 関数 [SOAP]

SOAP 要求ヘッダの次のヘッダ・キーを返します。

構文

NEXT_SOAP_HEADER(*header-key* **)**

パラメータ

header-key　指定したヘッダ・エントリの最上位 XML 要素の XML ローカル名。

備考

header-key に NULL を指定すると、この関数は SOAP ヘッダで見つかった最初のヘッダ・エントリのヘッダ・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]」 250 ページ
- 「sa_set_soap_header システム・プロシージャ」 959 ページ
- 「SOAP ヘッダの使用」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、SOAP ヘッダで最初に見つかるヘッダ・キーを取得します。

```
SET header_key = NEXT_SOAP_HEADER( NULL );
```

NOW 関数 [日付と時刻]

現在の年、月、日、時、分、秒、秒以下を返します。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(*)

備考

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

参照

- 「GETDATE 関数 [日付と時刻]」 170 ページ
- 「CURRENT TIMESTAMP 特別値」 31 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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

expression-1　比較される式。

expression-2　比較される式。

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」 17 ページ

標準と互換性

- **SQL/2003**　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「使用法」で説明します)、代わりに「ROW_NUMBER 関数 [その他]」 242 ページを使用します。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。

構文

NUMBER(*)

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。

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、クエリ式 (UNION、EXCEPT、INTERSECT)、抽出テーブルで 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 (「INSERT 文」 590 ページを参照してください)。ただし、連続したプライマリ・キーの生成には、AUTOINCREMENT 句の使用が好ましい方法です。

AUTOINCREMENT 句の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。

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」 23 ページ
- 「LOCATE 関数 [文字列]」 195 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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' );
```

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。PERCENT_RANK 関数は、0 から 1 の 10 進数値を返します。

構文

PERCENT_RANK( ) OVER (*window-spec* **)**

window-spec : 下の「備考」を参照

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」 745 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 - 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]」 135 ページ
- 「DENSE_RANK 関数 [ランキング]」 152 ページ
- 「RANK 関数 [ランキング]」 222 ページ

標準と互換性

- **SQL/2003** SQL／OLAP 機能 T612

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。結果セットは、小数のパーセンテージを使用して降順にランキングされ、性別によって分割されます。

```
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(*)

標準と互換性

- **SQL/2003**　ベンダ拡張。

備考

この関数は DOUBLE 値を返します。

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

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。

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。これらの値は、指定されたカーソルをオプティマイザ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。

参照

- 「EXPLANATION 関数 [その他]」 164 ページ
- 「GRAPHICAL_PLAN 関数 [その他]」 171 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、SELECT 文を文字列パラメータとして渡し、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' );
```

Interactive SQL では、SQL 文のプランを [結果] ウィンドウ枠の [プラン] タブに表示できます。

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

numeric-expression-1　べき乗の底。

numeric-expression-2　指数。

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。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/2003**　ベンダ拡張。

例

次の文は、値 64 を返します。

```
SELECT POWER( 2, 6 );
```

PROPERTY 関数 [システム]

指定したサーバレベルのプロパティの値を文字列で返します。

構文

PROPERTY({ *property-id* | *property-name* } **)**

パラメータ

property-id　整数。サーバレベル・プロパティのプロパティ番号です。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。

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

- 「サーバ・レベルのプロパティ」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、現在のデータベース・サーバ名を返します。

```
SELECT PROPERTY( 'Name' );
```

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

property-id　整数。データベース・プロパティのプロパティ番号です。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。

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

- 「データベース・プロパティ」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

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_NAME 関数 [システム]

指定したプロパティ番号を持つプロパティの名前を返します。

構文

PROPERTY_NAME(*property-id* **)**

パラメータ

property-id　データベース・プロパティのプロパティ番号。

参照

- 「データベース・プロパティの概要」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、プロパティ番号 126 に対応するプロパティを返します。この文が表す実際のプロパティは、リリースごとに変わります。

```
SELECT PROPERTY_NAME( 126 );
```

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

property-name　プロパティ名。

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使用できる場合、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

- 「データベース・プロパティの概要」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

指定した日付式から、四半期を示す数を返します。

構文

QUARTER(*date-expression* **)**

パラメータ

date-expression 日付。

備考

四半期は次のとおりです。

Quarter	期間 (開始日と終了日を含む)
1	1 月 1 日ー 3 月 31 日
2	4 月 1 日ー 6 月 30 日
3	7 月 1 日ー 9 月 30 日
4	10 月 1 日ー 12 月 31 日

標準と互換性

- **SQL/2003** ベンダ拡張。

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

numeric-expression　度数。この角度をラジアンに変換します。

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/2003**　ベンダ拡張。

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 ー 1 の乱数を返します。

構文

RAND([*integer-expression*] **)**

パラメータ

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。

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 ー 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。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 度のみ実行します。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 - SQL の使用法』を参照してください。

標準と互換性

- **SQL/2003** ベンダ拡張。

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。同順の場合、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 : **下の「備考」を参照**

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」 745 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 - 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]」 135 ページ
- 「DENSE_RANK 関数 [ランキング]」 152 ページ
- 「PERCENT_RANK 関数 [ランキング]」 214 ページ

標準と互換性

- **SQL/2003** SQL／OLAP 機能 T612

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
...	...	...	...
Lynch	24903.000	UT	19

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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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 *independent-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、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「AVG 関数 [集合]」 107 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

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, ( YEAR( NOW() ) - 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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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 *dependent-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、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_INTERCEPT 関数 [集合]」 227 ページ

- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「AVG 関数 [集合]」 107 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

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。この関数は、値 49988.6232 を返します。

```
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* **)**

構文 2

REGR_COUNT(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

備考

この関数は結果として LONG 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」 745 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 - 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、「集合関数の数学上の式」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

参照

- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「COUNT 関数 [集合]」 130 ページ
- 「AVG 関数 [集合]」 107 ページ
- 「SUM 関数 [集合]」 266 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

例

次の例は、回帰直線の調整に使用された NULL 以外のペアの数を示す値を返します。この関数は、値 75 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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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 *dependent-expression* を表し、*y* は *independent-expression* を表します。

```
AVG( x ) - REGR_SLOPE( x, y ) * AVG( y )
```

実行される統計計算の詳細については、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「AVG 関数 [集合]」 107 ページ

標準と互換性

- **SQL/2003** コア SQL に含まれない SQL 基本機能 (T621)。

例

次の例は、値 4680.6094936855225 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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

例

次の例は、値 0.19379959710325653 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* **)**

構文 2

REGR_SLOPE(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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 *dependent-expression* を表し、*y* は *independent-expression* を表します。

COVAR_POP(*x*, *y*) / VAR_POP(*y*)

実行される統計計算の詳細については、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ
- 「COVAR_POP 関数 [集合]」 131 ページ
- 「VAR_POP 関数 [集合]」 277 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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 *dependent-expression* を表し、*y* は *independent-expression* を表します。

```
REGR_COUNT( x, y ) * VAR_POP( x )
```

実行される統計計算の詳細については、「集合関数の数学上の式」 『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」 226 ページ

- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ
- 「VAR_POP 関数 [集合]」 277 ページ

標準と互換性

- **SQL/2003** コア SQL に含まれない SQL 基本機能 (T621)。

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結果の合計を返します。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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。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 *dependent-expression* を表し、*y* は *independent-expression* を表します。

```
REGR_COUNT( x, y ) * COVAR_POP( x, y )
```

実行される統計計算の詳細については、「集合関数の数学上の式」 『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SYY 関数 [集合]」 233 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

例

次の例は、値 5533938.004400015 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 : **下の「備考」の構文 2 の説明を参照**

パラメータ

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。

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。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 *dependent-expression* を表し、*y* は *independent-expression* を表します。

```
REGR_COUNT( x, y ) * VAR_POP( y )
```

実行される統計計算の詳細については、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_INTERCEPT 関数 [集合]」 227 ページ
- 「REGR_COUNT 関数 [集合]」 226 ページ
- 「REGR_AVGX 関数 [集合]」 223 ページ
- 「REGR_AVGY 関数 [集合]」 225 ページ
- 「REGR_SLOPE 関数 [集合]」 229 ページ
- 「REGR_SXX 関数 [集合]」 231 ページ
- 「REGR_SXY 関数 [集合]」 232 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

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* **)**

パラメータ

dividend　被除数 (分数の分子)。

divisor　除数 (分数の分母)。

備考

または、MOD 関数も使用できます。

参照

- 「MOD 関数 [数値]」 202 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。

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。

または、REPLICATE 関数も使用できます。

この関数は NCHAR の入力または 出力をサポートしています。

参照

- 「REPLICATE 関数 [文字列]」 237 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。

備考

この関数はすべてのオカレンスを置換します。

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

- 「SUBSTRING 関数 [文字列]」 264 ページ
- 「CHARINDEX 関数 [文字列]」 117 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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%';
```

LIST 関数にカンマ以外の区切り記号を使用する場合は、次のようになります。

```
SELECT REPLACE( LIST( table_id ), ',', '--')
FROM  SYS.SYSTAB
WHERE table_id <= 5;
```

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

string-expression　繰り返される文字列。

integer-expression　文字列を繰り返す回数。

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。

または、REPEAT 関数も使用できます。

この関数は NCHAR の入力または 出力をサポートしています。

参照

- 「REPEAT 関数 [文字列]」 235 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

string-expression　逆転される文字列。

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」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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。

構文

REWRITE(*select-statement* **[, 'ANSI'])**

パラメータ

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。

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理解でき、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がないかどうかを調べることができます。

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る

参照

- 「セマンティック・クエリ変形」 『SQL Anywhere サーバ - SQL の使用法』

- 「extended_join_syntax オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「Transact-SQL の外部ジョイン (*= or =*)」『SQL Anywhere サーバ - SQL の使用法』
- 「キー・ジョイン」『SQL Anywhere サーバ - SQL の使用法』
- 「ナチュラル・ジョイン」『SQL Anywhere サーバ - 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 - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

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.EmployeeID, e.GivenName
 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

string-expression 左側をトランケートされる文字列。

integer-expression　返される文字列の末尾の文字数。

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、戻り値は文字長のセマンティックの観点から記述されます。

参照

- 「LEFT 関数 [文字列]」 191 ページ
- 「国際言語と文字セット」 『SQL Anywhere サーバ - データベース管理』
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。

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]」 272 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 123.200 を返します。

```
SELECT ROUND( 123.234, 1 );
```

ROWID 関数 [その他]

テーブル内のローを一意に識別する符号なし 64 ビット値を返します。

構文

ROWID(*correlation-name* **)**

パラメータ

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 (基礎のプロキシ・サーバ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を参照しないでください。

備考

符号なし 64 ビット値 (BIGINT) という特定の相関に関連づけた表紙のロー識別子を返します。

データベースで実行される多様な演算によって、テーブルのロー識別子が変更される可能性があるため、クエリの間に関数が返す値は、定数とはかぎりません。特に、REORGANIZE TABLE 文は、ロー識別子が変更される可能性があります。さらに、ローを削除した後はロー識別子を減らすことができます。通常は ROWID 関数の使用は避ける必要があるため、代わりにプライマリ・キー値で取得される値を使用します。ROWID は診断する状況の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。さらに、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*) は 64 ビット UNSIGNED INTEGER 値にキャストします。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。*リテラル* の値が無効なロー識別子の場合、比較述部は FALSE に評価されます。

テーブルでもカラムでも、ROWID 関数は CHECK 定数内で使用できません。また、計算済みカラムの COMPUTE 式にも使用できません。

参照

- 「ROW_NUMBER 関数 [その他]」 242 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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';
```

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 : 下の「備考」を参照

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」 745 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 - SQL の使用法』を参照してください。

参照

- 「NUMBER 関数 [その他]」 212 ページ
- 「ROWID 関数 [その他]」 241 ページ

標準と互換性

- **SQL/2003** SQL／OLAP 機能 T612

例

次の例は、ニューヨークとユタの各従業員にユニークなロー番号を提供する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。

```
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

Surname	Salary	State	Rank
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

RTRIM 関数 [文字列]

後続ブランクが削除された文字列を返します。

構文

RTRIM(*string-expression* **)**

パラメータ

string-expression　削除される文字列。

備考

結果の実際の長さは、式の長さから、削除される文字数を引いた値です。すべての文字を削除すると、結果は空の文字列です。

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]」 272 ページ
- 「LTRIM 関数 [文字列]」 198 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

SQL/2003 規格で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数から指定されます。

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]

指定した日付の秒を返します。

構文

SECOND(*datetime-expression* **)**

パラメータ

datetime-expression　日時の値。

備考

指定した日時の秒に相当する 0 ー 59 の数を返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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]

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの秒数を返します。

  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。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つのタイムスタンプの間の秒数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定したタイムスタンプに、指定した整数の秒数が加算されます。代わりに、DATEADD 関数を使用します。

構文 1：整数

SECONDS([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

SECONDS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する秒数。*integer-expression* が負の場合、日時の値から適切な分数が引かれます。整数式を指定する場合は、*datetime-expression* を datetime データ型として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]」 115 ページ
- 「DATEADD 関数 [日付と時刻]」 137 ページ
- 「DATEDIFF 関数 [日付と時刻]」 138 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 14400 を返します。これは、2 番目のタイムスタンプが、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、日時 1999-05-12 21:05:12.0 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

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。

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]」 168 ページ
- 「SET_BITS 関数 [集合]」 246 ページ
- 「INTEGER データ型」 60 ページ
- 「ビット処理演算子」 13 ページ
- 「sa_get_bits システム・プロシージャ」 900 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 );
```

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

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。

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」 13 ページ
- 「GET_BIT 関数 [ビット配列]」 168 ページ
- 「SET_BIT 関数 [ビット配列]」 245 ページ
- 「sa_get_bits システム・プロシージャ」 900 ページ

標準と互換性

SQL/2003　ベンダ拡張。

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;
```

SHORT_PLAN 関数 [その他]

SQL 文の Ultra Light プランの最適化方法を短い記述の文字列で返します。この記述は、EXPLANATION 関数が返す記述と同じです。

構文

SHORT_PLAN(*string-expression* **)**

備考

クエリによっては、SQL Anywhere に選択したプランと Ultra Light の実行プランが異なる場合があります。

パラメータ

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。

参照

- 「PLAN 関数 [その他]」 216 ページ
- 「EXPLANATION 関数 [その他]」 164 ページ
- 「GRAPHICAL_PLAN 関数 [その他]」 171 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、SELECT 文を文字列パラメータとして渡し、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' );
```

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。

Interactive SQL では、SQL 文のプランを [結果] ウィンドウ枠の [プラン] タブに表示できます。

SIGN 関数 [数値]

数の符号を返します。

構文

SIGN(*numeric-expression* **)**

パラメータ

numeric-expression 符号が返される数。

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/2003** ベンダ拡張。

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

string-expression-1　比較される最初の文字列。

string-expression-2　比較される 2 番目の文字列。

備考

SIMILAR 関数は、2 つの文字列の類似性を表す 0 ー 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。テーブルをそのテーブル自体とジョインし、90% より大きく、100% 未満の類似性をすべてレポートします。

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」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

numeric-expression　角度 (ラジアン)。

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値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]」 105 ページ
- 「COS 関数 [数値]」 128 ページ
- 「COT 関数 [数値]」 129 ページ
- 「TAN 関数 [数値]」 267 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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・エントリまたは SOAP 要求のヘッダ・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

header-key VARCHAR パラメータには、特定の SOAP ヘッダ・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 パラメータは、同じ名前の SOAP ヘッダ・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・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 パラメータは、次に示すヘッダ・エントリ要素内にある任意の属性ノードです。

- **@namespace** 特定のヘッダ・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・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・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・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・エントリの受信者を示す SOAP 1.1 ヘッダ・エントリ属性。

備考

この関数は、単一のパラメータ *header-key* を使用してヘッダ・エントリを返すときに使用します。ヘッダ・エントリは、SOAP ヘッダ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* パラメータを指定して、ヘッダ・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・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* のヘッダが存在しない場合、戻り値は NULL です。

参照

- 「NEXT_SOAP_HEADER 関数 [SOAP]」 210 ページ
- 「sa_set_soap_header システム・プロシージャ」 959 ページ
- 「SOAP ヘッダの使用」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。このパラメータは、Adaptive Server Enterprise の照合にのみ適用され、対応する照合 ID によって参照できます。

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・ユニコード・マルチ言語です。

有効な照合は、次のとおりです。

- SQL Anywhere は照合をサポートします。サポートされている照合については、dbinit -l を実行して参照してください。対応するラベルとともに表示されます。
- 次の表に Adaptive Server Enterprise の照合を示します。

説明	照合名	照合 ID
デフォルト・ユニコード・マルチ言語	default	0
CP 850 Alternative：アクセントなし	altnoacc	39
CP 850 Alternative：小文字先頭	altdict	45
CP 850 西ヨーロッパ言語：大文字と小文字の区別なし、環境設定	altnocsp	46
CP 850 スカンジナビア語辞書	scandict	47
CP 850 スカンジナビア語辞書：大文字と小文字の区別なし、環境設定	scannocp	48
GB Pinyin	gbpinyin	なし
バイナリ・ソート	binary	50
ラテン語 -1 英語、フランス語、ドイツ語辞書	dict	51
ラテン語 -1 英語、フランス語、ドイツ語の大文字と小文字の区別なし	nocase	52
ラテン語 -1 英語、フランス語、ドイツ語の大文字と小文字の区別なし、環境設定	nocasep	53
ラテン語 -1 英語、フランス語、ドイツ語のアクセントなし	noaccent	54
ラテン語 -1 スペイン語辞書	espdict	55
ラテン語 -1 スペイン語の大文字と小文字の区別なし	espnocs	56
ラテン語 -1 スペイン語のアクセントなし	espnoac	57
ISO 8859-5 ロシア語辞書	rusdict	58
ISO 8859-5 ロシア語の大文字と小文字の区別なし	rusnocs	59

説明	照合名	照合 ID
ISO 8859-5 キリル語辞書	cyrdict	63
ISO 8859-5 キリル語の大文字と小文字の区別なし	cyrnocs	64
ISO 8859-7 ギリシャ語辞書	elldict	65
ISO 8859-2 ハンガリー語辞書	hundict	69
ISO 8859-2 ハンガリー語のアクセントなし	hunnoac	70
ISO 8859-2 ハンガリー語の大文字と小文字の区別なし	hunnocs	71
ISO 8859-5 トルコ語辞書	turdict	72
ISO 8859-5 トルコ語のアクセントなし	turnoac	73
ISO 8859-5 トルコ語の大文字と小文字の区別なし	turnocs	74
CP 874 (TIS 620) タイ語辞書	thaidict	1
ISO 14651 順序付け標準	14651	22
シフト JIS バイナリ順	sjisbin	179
ユニコード UTF-8 バイナリ・ソート	utf8bin	24
EUC JIS バイナリ順	eucjisbn	192
GB2312 バイナリ順	gb2312bn	137
CP932 MS バイナリ順	cp932bin	129
Big5 バイナリ順	big5bin	194
EUC KSC バイナリ順	euckscbn	161

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。「照合の適合化オプション」 382 ページを参照してください。

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。

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。

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。

照合の適合化に関して言えば、ソートキーを作成する場合は一般に完全な区別が意図されるため、UCA 以外の照合を指定すると、適用されるデフォルトの適合化は case=Respect を指定したのと同等の適合化になります。たとえば、次の 2 つの文は同じです。

```
SELECT SORTKEY( 'abc', '1252LATIN1' );
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

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)' );
```

適合化オプションを指定せずに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
バージョン 10.0.0 より前の SQL Anywhere で作成されたソート・キー値は、バージョン 10.0.0 以降で作成される値とは異なります。10.0.0 より前のデータベース内にソート・キー値が格納されている場合、特に、ソート・キー値の比較がアプリケーションで必要な場合、問題が発生する可能性があります。バージョン 10.0.0 より前の SQL Anywhere で生成されたデータベースでは、ソート・キー値を再生成する必要があります。

参照

- 「sort_collation オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「COMPARE 関数 [文字列]」 119 ページ
- 「国際言語と文字セット」 『SQL Anywhere サーバ - データベース管理』
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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 (ラテン語 -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

string-expression 評価される文字列。

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、apple 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。

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」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

integer-expression　返すスペースの数。

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」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

sql-statement-string　関数がダイアレクトの判断に使用する SQL 文。

参照

- 「TRANSACTSQL 関数 [その他]」 271 ページ
- 「WATCOMSQL 関数 [その他]」 280 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

sql-standard-string　準拠性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: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。

参照

- 「sql_flagger_error_level オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「SQL プリプロセッサ」 『SQL Anywhere サーバ - プログラミング』

- 「sa_ansi_standard_packages システム・プロシージャ」 870 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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 '先頭' 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 SQL 文の例のように、使用できない拡張機能が検出されると、'00000' が返されます。

```
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

numeric-expression 平方根が計算される数。

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/2003** ベンダ拡張。

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。「STDDEV_SAMP 関数 [集合]」 260 ページを参照してください。

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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。

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。

$s = [(1/N) * SUM(x_i - mean(x))^2]^{1/2}$

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、「集合関数の数学上の式」『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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。

$s = [(1/(N - 1)) * SUM(x_i - mean(x))^2]^{1/2}$

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、「集合関数の数学上の式」 『SQL Anywhere サーバ - 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」 745 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 - SQL の使用法』を参照してください。

参照

- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能 (T621)。

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

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。

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
サポートされる最大長は 128 です。1 ー 128 の範囲にない長さでは結果が NULL になります。

参照

- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、6 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 );
```

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

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。

備考

数値または日付をパラメータ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が NULL の場合、STRING は NULL を返します。NULL でないパラメータが存在すると、NULL パラメータ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」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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。

新しいデータベースでは不要
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 データ型はネイティブ・データ型であり、SQL Anywhere が必要に応じて変換を実行します。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」 75 ページを参照してください。

構文

STRTOUUID(*string-expression* **)**

パラメータ

string-expression　フォーマットが *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* の文字列。

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。

参照

- 「UUIDTOSTR 関数 [文字列]」 276 ページ
- 「NEWID 関数 [その他]」 206 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

```
CREATE TABLE T1 (
  pk UNIQUEIDENTIFIER PRIMARY KEY, c1 INT );
INSERT INTO T1 ( pk, c1 )
VALUES ( STRTOUUID('12345678-1234-5678-9012-123456789012'), 1 );
```

STUFF 関数 [文字列]

1 つの文字列から指定した文字数を削除して、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

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。

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]」 185 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

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。

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/2003 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	文字列の先頭文字の位置を 1 とします。負またはゼロ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たように扱われます。	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、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。	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。

SUBSTRING 関数では、開始オフセットを正でない値に設定したり負の長さを指定したりしないことをおすすめします。可能なかぎり、SUBSTRING 関数の代わりに LEFT 関数または RIGHT 関数を使用してください。

この関数は NCHAR の入力または 出力をサポートしています。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、戻り値は文字長のセマンティックの観点から記述されます。

参照

- 「BYTE_SUBSTR 関数 [文字列]」 114 ページ
- 「ansi_substring オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「LEFT 関数 [文字列]」 191 ページ
- 「RIGHT 関数 [文字列]」 239 ページ
- 「CHARINDEX 関数 [文字列]」 117 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の表に、SUBSTRING 関数を SELECT 文で使用して、ansi_substring オプションを On と Off に設定したときに返す値を示します。

例	On に設定された ansi_substring の結果	Off に設定された ansi_substring の結果
SUBSTRING('front yard', 1, 4)	fron	fron

例	On に設定された ansi_substring の結果	Off に設定された ansi_substring の結果
SUBSTRING('back yard', 6, 4)	yard	yard
SUBSTR('abcdefgh', 0, -2)	エラーを返します	gh
SUBSTR('abcdefgh', -2, 2)	空の文字列を返します	gh
SUBSTR('abcdefgh', 2, -2)	エラーを返します	ab
SUBSTR('abcdefgh', 2, -4)	エラーを返します	ab
SUBSTR('abcdefgh', 2, -1)	エラーを返します	b

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(*expression* | **DISTINCT** *expression* **)**

構文 2

SUM(*expression* **) OVER (** *window-spec* **)**

window-spec : 下の「備考」の構文 2 の説明を参照

パラメータ

expression　合計するオブジェクト。通常はカラム名です。

DISTINCT 式　入力の *expression* で一意の値の合計を計算します。

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」 745 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 - SQL の使用法』を参照してください。

参照

- 「COUNT 関数 [集合]」 130 ページ
- 「AVG 関数 [集合]」 107 ページ

標準と互換性

- **SQL/2003**　コア機能。構文 2 は T611 です。

例

次の文は、値 3749146.740 を返します。

```
SELECT SUM( Salary )
FROM Employees;
```

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

numeric-expression　角度 (ラジアン)。

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]」 128 ページ
- 「SIN 関数 [数値]」 249 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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]

指定したテキスト・カラムの最初のページへの 16 バイトのバイナリ・ポインタを返します。

構文

TEXTPTR(*column-name* **)**

パラメータ

column-name　テキスト・カラムの名前。

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/2003**　ベンダ拡張。

例

TEXTPTR を使用して、作者の blurbs テーブルで au_id 486-29-1786 と対応するテキスト・カラムを見つけて、コピーします。

テキスト・ポインタは、ローカル変数 @val の中に置かれ、readtext コマンドのパラメータとして指定されます。readtext コマンドは、第 2 バイトから開始して 5 バイトを返します (オフセットは 1)。

```
DECLARE @val VARBINARY(16)
SELECT @val = TEXTPTR(copy)
FROM blurbs
WHERE au_id = "486-29-1786"
READTEXT blurbs.copy @val 1 5 ;
```

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

string-expression　変換される文字列。

source-charset-name　文字列の文字セット。

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]」 133 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 - データベース管理』
- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」 122 ページ
- 「CSCONVERT 関数 [文字列]」 133 ページ

- 「NCHAR 関数 [文字列]」 205 ページ
- 「TO_NCHAR 関数 [文字列]」 269 ページ
- 「UNICODE 関数 [文字列]」 274 ページ
- 「UNISTR 関数 [文字列]」 274 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

string-expression　変換される文字列。

source-charset-name　文字列の文字セット。

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]」 133 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 );
```

参照

- 「推奨文字セットと照合」 『SQL Anywhere サーバ - データベース管理』
- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」 122 ページ
- 「CSCONVERT 関数 [文字列]」 133 ページ
- 「NCHAR 関数 [文字列]」 205 ページ
- 「TO_CHAR 関数 [文字列]」 268 ページ
- 「UNICODE 関数 [文字列]」 274 ページ
- 「UNISTR 関数 [文字列]」 274 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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' );
```

TODAY 関数 [日付と時刻]

現在の日付を返します。

構文

TODAY(*)

備考

従来の CURRENT DATE 関数の位置にこの構文を使用します。

標準と互換性

- **SQL/2003** ベンダ拡張。

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やトリガのトレースバックが含まれた文字列を返します。

構文

TRACEBACK(*)

備考

この関数は、プロシージャやトリガのデバッグに役立ちます。

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

例

traceback 関数を使用するには、プロシージャ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

logging_session_id　この INTEGER パラメータと *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。

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」 769 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

TRANSACTSQL 関数 [その他]

Watcom-SQL 文を取得し、Transact-SQL 表現で書き換えます。

構文

TRANSACTSQL(*sql-statement-string* **)**

パラメータ

sql-statement-string　関数がダイアレクトの判断に使用する SQL 文。

参照

- 「SQLDIALECT 関数 [その他]」 256 ページ
- 「WATCOMSQL 関数 [その他]」 280 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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;
```


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

string-expression　削除される文字列。

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]」 198 ページ
- 「RTRIM 関数 [文字列]」 243 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　TRIM 関数は SQL/2003 のコア機能です。

  SQL Anywhere では、SQL/2003 で定義されている追加のパラメータ *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/2003 規格で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数から指定されます。

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

numeric-expression　トランケートされる数。

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。

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 文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]」 240 ページ
- 「quoted_identifier オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

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 );
```

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。この関数は UPPER 関数と同じです。

構文

UCASE(*string-expression* **)**

パラメータ

string-expression 大文字に変換される文字列。

備考

UCASE 関数は UPPER 関数に似ています。

参照

- 「UPPER 関数 [文字列]」 275 ページ
- 「LCASE 関数 [文字列]」 190 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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ユニコード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

nchar-string-expression　最初の文字が整数に変換される NCHAR 文字列。

参照

- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」 122 ページ
- 「NCHAR 関数 [文字列]」 205 ページ
- 「TO_CHAR 関数 [文字列]」 268 ページ
- 「TO_NCHAR 関数 [文字列]」 269 ページ
- 「UNISTR 関数 [文字列]」 274 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、整数 65536 を返します。

```
SELECT UNICODE(UNISTR( '¥u010000data' ));
```

UNISTR 関数 [文字列]

文字とユニコード・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

string-expression　変換される文字列。

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ユニコード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 関数はユニコード文字と 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 には文字とユニコードのエスケープ・シーケンスが含まれます。ユニコードのエスケープ・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、個々の文字とユニコードのエスケープ・シーケンスを対応するユニコード文字に変換します。

6 桁のユニコードのエスケープ・シーケンスを使用する場合、値はユニコード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 桁のユニコード・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 つの隣接するユニコード・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 つのユニコード文字に結合されます。

参照

- 「CONNECTION_EXTENDED_PROPERTY 関数 [文字列]」 122 ページ
- 「NCHAR 関数 [文字列]」 205 ページ
- 「TO_CHAR 関数 [文字列]」 268 ページ
- 「TO_NCHAR 関数 [文字列]」 269 ページ
- 「UNICODE 関数 [文字列]」 274 ページ
- 「文字列」 8 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 をユニコード・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。この関数は UCASE 関数と同じです。

構文

UPPER(*string-expression* **)**

パラメータ

string-expression　大文字に変換される文字列。

備考

UCASE 関数は UPPER 関数に似ています。

参照

- 「UCASE 関数 [文字列]」 273 ページ
- 「LCASE 関数 [文字列]」 190 ページ
- 「LOWER 関数 [文字列]」 198 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 CHOCOLATE を返します。

```
SELECT UPPER( 'ChocoLate' );
```

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。

新しいデータベースでは不要
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 データ型はネイティブ・データ型であり、SQL Anywhere が必要に応じて変換を実行します。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」 75 ページを参照してください。

構文

UUIDTOSTR(*uuid-expression* **)**

パラメータ

uuid-expression　ユニークな識別子の値。

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]」 206 ページ
- 「STRTOUUID 関数 [文字列]」 263 ページ
- 「文字列関数」 99 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 : **下の「備考」の構文 2 の説明を参照**

パラメータ

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。

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」 745 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 - SQL の使用法』を参照してください。

参照

- 「集合関数」 93 ページ

標準と互換性

- **SQL/2003** コア SQL に含まれない SQL 基本機能 (T611)。

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 : 下の「備考」の構文 2 の説明を参照

パラメータ

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。

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」 745 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 - SQL の使用法』を参照してください。

参照

- 「集合関数」 93 ページ
- 「VARIANCE 関数 [集合]」 280 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能。VARIANCE 構文はベンダ拡張です。

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…
…	…	…	…

VARIANCE 関数 [集合]

VAR_SAMP のエイリアス。「VAR_SAMP 関数 [集合]」 278 ページを参照してください。

VAREXISTS 関数 [その他]

指定した名前でユーザ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

variable-name-string　文字列としてテストされる変数名。

参照

- 「CREATE VARIABLE 文」 480 ページ
- 「DECLARE 文」 488 ページ
- 「IF 文」 580 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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;
```

WATCOMSQL 関数 [その他]

Transact-SQL 文を取得し、Watcom-SQL 表現で書き換えます。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

sql-statement-string　関数がダイアレクトの判断に使用する SQL 文。

参照

- 「SQLDIALECT 関数 [その他]」 256 ページ
- 「TRANSACTSQL 関数 [その他]」 271 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、文字列 'SELECT empl_name AS EmployeeName FROM Employees' を返します。

```
SELECT WATCOMSQL( 'SELECT EmployeeName=empl_name FROM Employees' ) FROM dummy;
```

WEEKS 関数 [日付と時刻]

2 つの日付を指定すると、2 つの日付の間の週数を整数で返します。この目的で使用する場合は、「DATEDIFF 関数 [日付と時刻]」 138 ページ を代わりに使用することをおすすめします。

1 つの日付を指定すると、0000-02-29 からの週数を返します。

1 つの日付と整数を指定すると、指定した日付に、指定した整数の週数が加算されます。この目的で使用する場合は、「DATEADD 関数 [日付と時刻]」 137 ページ を代わりに使用することをおすすめします。

構文 1 は、整数を返します。構文 2 は、タイムスタンプを返します。

構文 1

WEEKS([*datetime-expression*,] *datetime-expression* **)**

構文 2

WEEKS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression 日付と時刻。

integer-expression *datetime-expression* に加算する週数。*integer-expression* が負の場合、日時の値から適切な週数が引かれます。*integer-expression* を指定する場合は、*datetime-expression* を datetime データ型として明示的にキャストしてください。

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]」 115 ページを参照してください。

備考

2 つの日付の間の週数は、日曜日の数で計算します。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 8 を返します。これは、2 番目の日付が、最初の日付の 8 週間後であることを示します。2 番目の形式 (DATEDIFF) の使用をおすすめします。

```
SELECT WEEKS( '1999-07-13 06:07:12',
  '1999-09-13 10:07:12' );
```



```
SELECT DATEDIFF( week,
  '1999-07-13 06:07:12',
  '1999-09-13 10:07:12' );
```

次の文は、値 104270 を返します。

```
SELECT WEEKS( '1998-07-13 06:07:12' );
```

次の文は、タイムスタンプ 1999-06-16 21:05:07.0 を返します。2 番目の形式 (DATEADD) の使用をおすすめします。

```
SELECT WEEKS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5);

SELECT DATEADD( week, 5, '1999-05-12 21:05:07' );
```

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(*value-expression* [**ORDER BY** *order-by-expression*] **)**

パラメータ

value-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。

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 - 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 - SQL の使用法』

標準と互換性

- SQL/XML ドラフト規格の一部。

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

xml-value　連結された XML 値。

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 - SQL の使用法』
- 「XMLFOREST 関数 [文字列]」 286 ページ
- 「文字列関数」 99 ページ

標準と互換性

- SQL/XML ドラフト規格の一部。

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

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。*

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' );
```

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 - 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]」 286 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 - SQL の使用法』
- 「XMLFOREST 関数 [文字列]」 286 ページ
- 「文字列関数」 99 ページ

標準と互換性

- SQL/XML ドラフト規格の一部。
- 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、ベンダ拡張です。

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 website</A> を返します。

```
SELECT XMLELEMENT(
  'A',
  XMLATTRIBUTES( 'http://www.ianywhere.com/'
    AS "HREF", '_top' AS "TARGET"),
  'iAnywhere website'
);
```

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

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。

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、NULL 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]」 284 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 - SQL の使用法』
- 「XMLELEMENT 関数 [文字列]」 284 ページ
- 「XMLCONCAT 関数 [文字列]」 283 ページ
- 「文字列関数」 99 ページ

標準と互換性

- SQL/XML ドラフト規格の一部。

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

xquery-constructor　XQuery コンストラクタ。XQuery コンストラクタは、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;
```

備考

XQuery 仕様で定義されている計算コンストラクタは、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 - 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 - SQL の使用法』
- 「文字列関数」 99 ページ

標準と互換性

- SQL/XML ドラフト規格の一部。

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]

タイムスタンプ値をパラメータとして受け取り、そのタイムスタンプで指定された年を返します。

構文

YEAR(*datetime-expression* **)**

パラメータ

datetime-expression　日付、時刻、またはタイムスタンプ。

備考

値は short 値として返されます。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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]

2 つの日付を指定すると、2 つの日付の間の年数を整数で返します。この目的で使用する場合は、「DATEDIFF 関数 [日付と時刻]」 138 ページ を代わりに使用することをおすすめします。

1 つの日付を指定すると、その年が返されます。この目的で使用する場合は、「DATEPART 関数 [日付と時刻]」 140 ページ を代わりに使用することをおすすめします。

1 つの日付と整数を指定すると、指定した日付に、指定した整数の年数が加算されます。この目的で使用する場合は、「DATEADD 関数 [日付と時刻]」 137 ページ を代わりに使用することをおすすめします。

構文 1

YEARS([*datetime-expression*,] *datetime-expression* **)**

構文 2

YEARS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する年数。*integer-expression* が負の場合、日時の値から適切な年数が引かれます。*integer-expression* を指定する場合は、*datetime-expression* を datetime データ型として明示的にキャストしてください。

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]」 115 ページを参照してください。

備考

YEARS の値を求めるには、2 つの日付の間に年の最初の日がいくつあるかを計算します。

構文 1 は、整数を返します。構文 2 は、タイムスタンプを返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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。値は -32768 ー 32767 の small integer です。

構文

YMD(
integer-expression1,
integer-expression2,
integer-expression3 **)**

パラメータ

integer-expression1　年。

integer-expression2　月の数。月が 1 ー 12 の値でない場合は、年が相応に調整されます。

integer-expression3　日の数。任意の整数を指定でき、日付が相応に調整されます。

標準と互換性

- **SQL/2003**　ベンダ拡張。

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は、日付が相応に調整されます。たとえば、次の文は値 2000-03-01 を返します。

```
SELECT YMD( 1999, 15, 1 );
```

第 4 章

SQL 文

目次

SQL 文リファレンスの使い方

この項では、SQL 文の説明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。

ここで説明する要素の詳細については、「識別子」 7 ページ、「SQL データ型」 47 ページ、「探索条件」 20 ページ、「SQL データ型」 47 ページ、「式」 15 ページ、または「文字列」 8 ページを参照してください。

- **column-name**
  カラム名を表す識別子。「識別子」 7 ページを参照してください。
- **condition**
  TRUE、FALSE、または UNKNOWN の評価を行う式。「真理値探索条件」 26 ページを参照してください。
- **connection-name**
  アクティブな接続の名前を表す文字列。「SQL Anywhere データベース接続の概要」 『SQL Anywhere サーバ - データベース管理』を参照してください。
- **data-type**
  記憶データ型。「SQL データ型」 47 ページを参照してください。
- **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。「式」 15 ページを参照してください。
- **file-name**
  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。「ホスト変数の使用」 『SQL Anywhere サーバ - プログラミング』を参照してください。
- **indicator-variable**
  通常のホスト変数の直後に置かれた **short int** 型の二次的なホスト変数。この変数の前にはコロンを付けてください。インジケータ変数は、データベースとの NULL 値の受け渡しに使用されます。「ホスト変数の使用」 『SQL Anywhere サーバ - プログラミング』を参照してください。
- **materialized-view-name**
  実体化ビュー (Materialized View) 名を表す識別子。「実体化ビュー (Materialized View) の編集」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- **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 ID を表す識別子。「所有権パーミッションの概要」 『SQL Anywhere サーバ - データベース管理』を参照してください。
- **query-block**
  クエリ・ブロックは、簡単なクエリ式、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。「エンティティと関係」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- **savepoint-name**
  セーブポイント名を表す識別子。「トランザクション内のセーブポイント」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- **search-condition**
  TRUE、FALSE、または UNKNOWN の評価を行う条件。「探索条件」 20 ページを参照してください。
- **special-value**
  「特別値」 30 ページで説明する特別値の 1 つ。
- **statement-label**
  ループまたは複合文のラベルを表す識別子。「制御文」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- **string-expression**
  文字列に解決される式。「式」 15 ページを参照してください。
- **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。「FROM 句」 552 ページと「キー・ジョイン」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- **table-name**
  テーブル名を表す識別子。「識別子」 7 ページを参照してください。
- **userid**
  ユーザ名を表す識別子。「識別子」 7 ページを参照してください。
- **variable-name**

変数名を表す識別子。「変数」 36 ページを参照してください。

- **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)。「識別子」 7 ページを参照してください。

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*

- **プレースホルダ**　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* ように斜体で表記します。

  ALTER TABLE [*owner.*]*table-name*

- **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。

  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、項目間を縦線で区切り、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

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、トリガ、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。場合によっては、Transact-SQL フォーマット以外のストアド・プロシージャ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/2003 標準に近い代わりの文が推奨される場合もあります。
- **[Mobile Link]** Mobile Link クライアント専用の文
- **[SQL Remote]** SQL Remote 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・プロシージャでも使用できる文であることを意味します。

ALLOCATE DESCRIPTOR 文 [ESQL]

この文は、SQL 記述子領域 (SQLDA) に使用する領域を割り付けます。

構文

ALLOCATE DESCRIPTOR *descriptor-name*
[**WITH MAX** { *integer* | *hostvar* }]

descriptor-name : *string*

パラメータ

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]」 486 ページ
- 「SQLDA (SQL descriptor area)」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003**　コア機能。

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

この文は、データベースをアップグレードするとき、データベースの jConnect サポートのオン／オフを切り替えるとき、データベースを調整するとき、トランザクション・ファイル名とミラー・ログ・ファイル名を変更するとき、またはミラー・サーバにデータベースの所有権の取得を強制するときに使用します。

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 PARALLEL READ
| RESTORE DEFAULT CALIBRATION
}

構文 3 - トランザクション名とミラー・ログ名の変更

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**
| ALTER DATABASE SET PARTNER FAILOVER }

パラメータ

PROCEDURE 句　データベースに含まれるすべての dbo 所有プロシージャと sys 所有プロシージャを削除して再作成します。

JCONNECT 句　Sybase jConnect JDBC ドライバからシステム・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。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 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・ログまたはトランザクション・ログ・ミラーのファイル名を変更します。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・ログまたはミラー・ログをオフにすることもできます。たとえば、ALTER DATABASE LOG OFF のように指定します。

KEY 句　トランザクション・ログまたはミラー・ログ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] 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・サーバとして動作しているデータベース・サーバに、データベースの所有権の取得を強制します。この文は、プライマリ・サーバ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。プロシージャまたはイベントから実行できます。「データベース・サーバの強制プライマリ・サーバ化」『SQL Anywhere サーバ - データベース管理』を参照してください。

SET PARTNER FAILOVER　プライマリ・サーバからミラー・サーバへのデータベース・ミラーリングのフェールオーバを起動します。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。結果として、文がプロシージャ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。「プライマリ・サーバのフェールオーバの起動」『SQL Anywhere サーバ - データベース管理』を参照してください。

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・プロシージャを強制的に再構築できます。

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。「データベースの再構築」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

アップグレード前のバックアップ
すべてのソフトウェアに共通することですが、データベースのバックアップを作成してからアップグレードすることをおすすめします。「バックアップとデータ・リカバリ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Sybase jConnect JDBC ドライバ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。「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 - プログラミング』を参照してください。

構文 2　構文 2 を使用すると、オプティマイザが使用する I/O コスト・モデルを再調整することもできます。この場合、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・サーバを他の用途に使用できません。また、コンピュータ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。
「DB_EXTENDED_PROPERTY 関数 [システム]」 144 ページを参照してください。

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・ログとミラー・ログの名前を変更できます。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・ログまたはミラー・ログを変更する場合は、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、トリガ、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 4　現在ミラーされていないデータベースに対して、または現在アクティブでこのサーバによって所有されてるデータベースに対して、ALTER DATABASE *dbname* 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・サーバがまだミラー・サーバに接続されている場合も、エラーになります。「データベース・ミラーリングの概要」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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 CE 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、実行するユーザには、-gk サーバ・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。

関連する動作

オートコミット。

参照

- 「CREATE DATABASE 文」 379 ページ
- 「アップグレード・ユーティリティ (dbupgrad)」 『SQL Anywhere サーバ - データベース管理』
- 「CREATE STATISTICS 文」 452 ページ
- 「トランザクション・ログ・ユーティリティ (dblog)」 『SQL Anywhere サーバ - データベース管理』
- 「DB_EXTENDED_PROPERTY 関数 [システム]」 144 ページ
- 「-gu サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

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 *newdemo.log* に設定します。

```
ALTER DATABASE 'demo.db'
  ALTER LOG ON 'newdemo.log';
```

ALTER DBSPACE 文

この文は、DB 領域またはトランザクション・ログ用に領域を割り付ける場合、または DB 領域ファイルの名前変更時や移動時に使用します。

構文

ALTER DBSPACE { *dbspace-name* | **TRANSLOG** | **TEMPORARY** }
{ **ADD** *number* [**PAGES** | **KB** | **MB** | **GB** | **TB**]
| **RENAME** *file-name-string* }

パラメータ

TRANSLOG この特別な 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。

TEMPORARY 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 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。「事前定義の DB 領域」『SQL Anywhere サーバ - データベース管理』を参照してください。

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 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。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 句で指定したの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。

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・データベース (ルート・ファイルとも呼ばれます) または追加の DB 領域を修正します。データベースの DB 領域名は、ISYSFILE システム・テーブルに保持されます。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・データベースを起動すると、起動ラインまたは ODBC データ・ソースの記述が、SQL Anywhere にメイン・データベース・ファイルの場所を知らせます。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」 388 ページ
- 「default_dbspace オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「データベースの編集」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

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

この文は、ユーザ定義のドメインまたはデータ型の名前を変更するために使用します。

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。

> **注意**
> ユーザ定義ドメインまたはデータ型を参照するプロシージャ、トリガ、ビュー、またはイベントは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

- 「ISYSFILE システム・テーブル」 754 ページ
- 「CREATE DOMAIN 文」 392 ページ
- 「ドメイン」 78 ページ
- 「ドメインの使い方」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

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

この文は、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・ハンドラの定義を変更するとき、またはスケジュールされたアクションの定義を変更するときに使用します。

構文

ALTER EVENT *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** | **"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*

パラメータ

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。イベント・タイプの説明については、「システム・イベントの概要」『SQL Anywhere サーバ - データベース管理』を参照してください。

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。パラメータの説明については、「CREATE EVENT 文」 397 ページを参照してください。

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。

- ALTER EVENT を使って開発時にイベント・ハンドラを変更できます。
- 開発段階でトリガ条件やスケジュールを指定せずにイベント・ハンドラを定義、テストし、イベント・ハンドラの完成後に ALTER EVENT を使って実行条件を追加することができます。
- イベントを無効にすることにより、イベント・ハンドラを一時的に無効にできます。

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

- 「BEGIN 文」 356 ページ
- 「CREATE EVENT 文」 397 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

ALTER FUNCTION 文

この文を使用して、関数を修正します。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 [*owner*.]*function-name function-definition*

function-definition : CREATE FUNCTION syntax

構文 2

ALTER FUNCTION [*owner*.]*function-name* **SET HIDDEN**

備考

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。CREATE FUNCTION 文のどちらのバージョンも変更できます。

関数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。DROP FUNCTION と CREATE FUNCTION を実行した場合は、EXECUTE パーミッションを再び割り当ててください。

構文 2　SET HIDDEN を使用して、関連する関数定義にスクランブルをかけ、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。

> *この設定は、元に戻せません。*元のソースが再び必要な場合は、データベース外で保存してください。

SET HIDDEN を使用すると、デバッガを使用したデバッグでも、プロシージャ・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。

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。

標準と互換性

- **SQL/2003**　ベンダ拡張。

参照

- 「CREATE FUNCTION 文」 408 ページ
- 「ALTER PROCEDURE 文」 319 ページ
- 「DROP 文」 512 ページ
- 「プロシージャ、関数、トリガ、ビューの内容を隠す」 『SQL Anywhere サーバ - SQL の使用法』

ALTER INDEX 文

この文を使用して、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

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。

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実体化ビュー (Materialized View) 上で 1 つのインデックスのみ、クラスタ化できます。

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。

パーミッション

テーブル所有者であるか、テーブルまたは実体化ビュー (Materialized View) に対する REFERENCES パーミッションまたは DBA 権限が必要です。

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。「スナップショット・アイソレーション」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

関連する動作

オートコミット。Interactive SQL の [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。

参照

- 「CREATE INDEX 文」 414 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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 MATERIALIZED VIEW 文

この文を使用して、実体化ビュー (Materialized View) 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**
}

percent-free-space : integer

パラメータ

SET HIDDEN 句 SET HIDDEN 句は、実体化ビュー (Materialized View) 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。*詳細については、「実体化ビュー (Materialized View) を隠す」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

ENABLE 句 ENABLE 句は、実体化ビュー (Materialized View) を有効にして、データベース・サーバから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。データを含む実体化ビュー (Materialized View) を初期化するには、この句を使用した後に REFRESH MATERIALIZED VIEW 文を実行します。

DISABLE 句 DISABLE 句は、データベース・サーバから実体化ビュー (Materialized View) を使用できないようにします。実体化ビュー (Materialized View) を無効にすると、データベース・サーバはビューのデータとすべてのインデックスを削除します。ビューを改めて有効にした後に、インデックスを再構築し、ビューを更新する必要があります。

{ ENABLE | DISABLE } USE IN OPTIMIZATION 句 この句は、オプティマイザから実体化ビュー (Materialized View) 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、実体化ビュー (Materialized View) 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。「オプティマイザによる実体化ビュー (Materialized View) の使用の有効化と無効化」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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 ー 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 の場合は 200 バイト、2 KB の場合は 100 バイト)。

DROP PCTFREE 句 現在の実体化ビュー (Materialized View) 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　実体化ビュー (Materialized View) 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、実体化ビュー (Materialized View) データは作成時に暗号化されません。実体化ビュー (Materialized View) を暗号化するには、ENCRYPTED を指定します。実体化ビュー (Materialized View) を復号化するには、NOT ENCRYPTED を指定します。

備考

実体化ビュー (Materialized View) を無効にすると、そのビューのすべてのインデックスが削除されます。必要に応じてビューを改めて有効にする場合は、再作成する必要があります。

データを含む実体化ビュー (Materialized View) を設定するには、実体化ビュー (Materialized View) を有効にしてから (ENABLE 句)、REFRESH MATERIALIZED VIEW 文を実行します。

実体化ビュー (Materialized View) を無効にした後は (DISABLE 句)、データベース・サーバがクエリに応答するときに実体化ビュー (Materialized View) を使用できなくなります。実体化ビュー (Materialized View) に依存するすべてのビューも無効になります。実体化ビュー (Materialized View) のデータは破棄されますが、ビューの定義はデータベースに残ります。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。「実体化ビュー (Materialized View) の有効化と無効化」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

実体化ビュー (Materialized View) を暗号化するには (ENCRYPT 句)、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実体化ビュー (Materialized View) が暗号化されます。「実体化ビュー (Materialized View) の暗号化と復号化」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

パーミッション

実体化ビュー (Materialized View) 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 MATERIALIZED VIEW 文」 420 ページ
- 「REFRESH MATERIALIZED VIEW 文」 640 ページ
- 「DROP 文」 512 ページ
- 「実体化ビュー (Materialized View) の編集」 『SQL Anywhere サーバ - SQL の使用法』
- 「ビューの依存性」 『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE VIEW 文」 482 ページ
- 「ALTER VIEW 文」 346 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、EmployeeSalary 実体化ビュー (Materialized View) を暗号化します。


```
ALTER MATERIALIZED VIEW EmployeeSalary
ENCRYPTED;
```

ALTER PROCEDURE 文

この文は、プロシージャを修正したり、Sybase Replication Server でのレプリケーションのプロシージャを有効にしたり無効にするために使用します。新しいプロシージャ全体を ALTER PROCEDURE 文にインクルードします。

PROC は PROCEDURE の同義語として使用できます。

構文 1

ALTER PROCEDURE [*owner*.]*procedure-name procedure-definition*

procedure-definition : CREATE PROCEDURE syntax

構文 2

ALTER PROCEDURE [*owner*.]*procedure-name*
REPLICATE { **ON** | **OFF** }

構文 3

ALTER PROCEDURE [*owner*.]*procedure-name* **SET HIDDEN**

備考

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。CREATE PROCEDURE 文のどちらのバージョンも変更できます。

プロシージャ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。DROP PROCEDURE と CREATE PROCEDURE を実行した場合は、EXECUTE パーミッションを再び割り当ててください。

構文 2　プロシージャが Sybase Replication Server によって他のサイトにレプリケートされる場合は、プロシージャに REPLICATE ON を設定します。

構文 3　SET HIDDEN を使用して、関連するプロシージャの定義にスクランブルをかけ、解読できないようにします。このプロシージャはアンロードして、他のデータベースに再ロードできます。

> *この設定は、元に戻せません。*元のソースが再び必要な場合は、データベース外で保存してください。

SET HIDDEN を使用すると、デバッガを使用したデバッグでも、プロシージャ・プロファイリングによっても、プロシージャ定義は表示されません。

構文 2 と構文 1 を組み合わせることはできません。構文 3 は、構文 1 とも構文 2 とも組み合わせることはできません。

パーミッション

プロシージャの所有者であるか、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」 423 ページ
- 「ALTER FUNCTION 文」 313 ページ
- 「DROP 文」 512 ページ
- 「プロシージャ、関数、トリガ、ビューの内容を隠す」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]

この文を使用して、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。

ALTER PUBLICATION 文は、データベース内でパブリケーションを変更します。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。

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]」 436 ページ
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」 518 ページ
- SQL Anywhere Mobile Link クライアント: 「データのパブリッシュ」 『Mobile Link - クライアント管理』
- Ultra Light Mobile Link クライアント : 「Ultra Light での同期の設計」 『Mobile Link - クライアント管理』
- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 332 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 455 ページ
- 「ISYSSYNC システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、Customer 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]

この文は、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、パブリッシャのメッセージ・システム、またはパブリッシャのアドレスの変更に使用します。

構文

ALTER REMOTE MESSAGE TYPE *message-system*
ADDRESS *address*

message-system: **FILE** | **FTP** | **MAPI** | **SMTP** | **VIM**

address: *string*

パラメータ

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ディレクトリを指定します。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。

> **注意**
> VIM および MAPI のサポートは廃止されました。

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

- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」 440 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、FILE メッセージ・リンクのパブリッシャのアドレスを new_addr に変更します。

```
ALTER REMOTE MESSAGE TYPE file
ADDRESS 'new_addr';
```

ALTER SERVER 文

この文は、リモート・サーバの属性を変更するために使用します。

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 :
SAJDBC | **ASEJDBC** | **SAODBC** | **ASEODBC**
| **DB2ODBC** | **MSSODBC** | **ORAODBC** | **ODBC**

connection-info :
computer-name:*port-number*[/*dbname*] | *data-source-name*

パラメータ

CLASS 句 CLASS 句は、サーバのクラスを変更するために指定されます。

サーバ・クラスの詳細とサーバの設定方法については、「リモート・データ・アクセスのサーバ・クラス」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

USING 句 USING 句は、サーバの接続情報を変更するために指定されます。*connection-info* の詳細については、「CREATE SERVER 文」 444 ページを参照してください。

CAPABILITY 句 CAPABILITY 句は、サーバの機能の ON と OFF を切り替えます。サーバ機能の情報は ISYSCAPABILITY システム・テーブルに格納されます。サーバ機能の名前は、システム・テーブル ISYSCAPABILITYNAME に保管されています。リモート・サーバからサーバへの接続が確立されるまでは、ISYSCAPABILITYNAME システム・テーブルにリモート・サーバのエントリは含まれていません。最初の接続時に、SQL Anywhere がデータベース・サーバに対して機能に関する問い合わせを行い、ISYSCAPABILITY テーブルにエントリを設定します。後続の接続では、このテーブルからデータベース・サーバの機能が取得されます。

通常、サーバ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
ユーザがリモート・サーバへの接続を作成した場合、リモート接続は、ユーザがローカル・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に DBA 権限が必要です。

備考

ALTER SERVER 文は、サーバの属性を変更します。変更された設定は、次回リモート・サーバに接続する際に有効になります。

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

- 「ISYSCAPABILITY システム・テーブル」 753 ページおよび 「ISYSCAPABILITYNAME システム・テーブル」 753 ページ
- 「CREATE SERVER 文」 444 ページおよび 「DROP SERVER 文」 520 ページ
- 「リモート・データ・アクセスのサーバ・クラス」 『SQL Anywhere サーバ - SQL の使用法』
- 「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 - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例は、Adaptive Server Entreprise サーバ ase_prod のサーバ・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 infodc の機能を変更します。

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF;
```

次の例は、リモート サーバ 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 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

この文を使用して Web サービスを変更します。

構文 1 - DISH サービス

ALTER SERVICE *service-name*
[**TYPE 'DISH'**]
[**GROUP** { *group-name* | **NULL** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** | **'XML'** | **NULL** }]
[*common-attributes*]

構文 2 - SOAP サービス

ALTER SERVICE *service-name*
[**TYPE 'SOAP'**]
[**DATATYPE** { **ON** | **OFF** | **IN** | **OUT** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** | **'XML'** | **NULL** }]
[*common-attributes*]
[**AS** *statement*]

構文 3 - その他のサービス

ALTER SERVICE *service-name*
[**TYPE** { **'RAW'** | **'HTML'** | **'XML'** }]
[**URL** [**PATH**] { **ON** | **OFF** | **ELEMENTS** }]
[*common-attributes*]
[**AS** { *statement* | **NULL** }]

common-attributes:
[**AUTHORIZATION** { **ON** | **OFF** }]
[**SECURE** { **ON** | **OFF** }]
[**USER** { *user-name* | **NULL** }]

パラメータ

service-name　変更するサービスを識別します。

service-type-string　サービスの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。

AUTHORIZATION 句　サービスに接続するときに、HTTP 基本認証を通じてユーザ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。認証が OFF の場合、AS 句が必要となり、USER 句によって 1 人のユーザが識別されます。そのユーザ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。

認証が ON の場合、すべてのユーザはユーザ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 句を使用しユーザ名またはグループ名を指定することにより、ユーザに対しサービスの使用許可を制限することもできます。ユーザ名が NULL の場合、既知のユーザはすべてサービスにアクセスできます。

デフォルト値は ON です。運用システムでは認証を ON にして実行すること、ユーザをグループに追加することによってサービス使用のパーミッションを付与することをおすすめします。

SECURE 句　安全化されていない接続を受け入れるかどうかを示します。ON は、HTTPS 接続のみを受け入れることを示します。HTTP ポートで受信されたサービス要求は、自動的に HTTPS

ポートにリダイレクトされます。OFF に設定すると、HTTP 接続と HTTPS 接続の両方を受け入れます。デフォルト値は OFF です。

USER 句　認証を無効にすると、このパラメータは必須となり、すべてのサービス要求を実行する際に使用されるユーザ ID が指定されます。認証を有効 (デフォルト) にすると、このオプション句は、サービスへのアクセスを許可されるユーザまたはグループを識別します。デフォルト値は NULL です。これによって、すべてのユーザにアクセスが付与されます。

URL 句　URI 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。OFF は、URI 要求のサービス名の後に何も指定する必要がないことを示します。ON は、URI の残りの部分が url という名前の変数の値として解釈されることを示します。ELEMENTS は、URI パスの残りの部分が、スラッシュで、最大 10 の要素からなるリストに分割されることを示します。値は、url という名前の変数と 1 から 10 までの数値サフィックスに割り当てられます。たとえば、最初の 3 つの変数名は、url1、url2、url3 です。指定される値の数が 10 より少ない場合、残りの変数は NULL に設定されます。サービス名が フォワード・スラッシュ / で終わる場合、URL を OFF に設定する必要があります。デフォルト値は OFF です。

GROUP 句　DISH サービスにだけ適用されます。DISH サービスがどの SOAP サービスを公開するかを制御する共通プレフィクスを指定します。たとえば、GROUP *xyz* を指定すると、SOAP サービス *xyz*/*aaaa*、*xyz*/*bbbb*、または *xyz*/*cccc* のみ公開され、*abc/aaaa* または *xyzaaaa* は公開されません。グループ名が指定されていない場合、DISH サービスはデータベース内のすべての SOAP サービスを公開します。SOAP サービスは、複数の DISH サービスによって公開される場合があります。サービス名で使用できる文字と同じ文字をグループ名に使用できます。

DATATYPE 句　SOAP サービスにだけ適用されます。すべての SOAP サービス・フォーマットでパラメータ入力または結果セットの出力にデータ入力をサポートするかどうかを制御します。サポートする場合、データ入力では SOAP ツールキットを使用してデータを解析し、適切な型にキャストします。パラメータのデータ型は、DISH サービスが生成する Web サービス定義言語 (WSDL: Web Service Definition Language) 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**　入力パラメータと結果セットの応答のデータ入力をサポートします。
- **OFF**　入力パラメータと結果セットの応答のデータ入力をサポートしません (デフォルト)。
- **IN**　入力パラメータのデータ入力のみをサポートします。
- **OUT**　結果セットの応答のデータ入力のみをサポートします。

FORMAT 句　DISH および SOAP サービスにだけ適用されます。.NET や Java JAX-RPC などのさまざまなタイプの SOAP クライアントと互換性のある出力フォーマットを生成します。SOAP サービスのフォーマットが指定されていない場合、フォーマットはサービスの DISH サービス宣言から継承されます。DISH サービスでもフォーマットが宣言されていない場合、デフォルトは、.NET クライアントと互換性のある DNET です。フォーマットを宣言しない SOAP サービスは、それぞれ異なる FORMAT タイプを持つ複数の DISH サービスを定義して、別のタイプの SOAP クライアントで使用できます。

statement　statement が NULL の場合、実行する statement を URI で指定する必要があります。そうしないと、指定された SQL 文しかサービスを使って実行されません。SOAP サービスには statement が必要ですが、DISH サービスには必要ありません。デフォルト値は NULL です。

運用システムで実行するすべてのサービスに statement を定義することを強くおすすめします。認証が有効になっている場合のみ、statement に NULL が指定できます。

フォーマット・タイプ

- **DNET**　.NET SOAP クライアントに使用する Microsoft DataSet フォーマット。DNET はデフォルトの FORMAT 値であり、バージョン 9.0.2 より前で使用できた唯一のフォーマットです。
- **CONCRETE**　プラットフォームに依存しない DataSet フォーマットであり、JAX-RPC などのクライアント、または返されるデータ構造のフォーマットに基づいてインタフェースを自動的に生成するクライアントで使用します。このフォーマット・タイプを指定すると、WSDL 内の ASADataset 要素が公開されます。この要素は、結果セットを、それぞれカラム要素の配列を含むローの配列から構成されるローセットの包含階層として記述します。
- **XML**　単純な XML 文字列フォーマット。XML パーサに渡すことのできる文字列として DataSet が返されます。このフォーマットは、SOAP クライアント間で最も移植が簡単です。

サービス・タイプ

- **RAW**　結果セットがフォーマットされずにクライアントに送られます。要求されたタグをプロシージャ内で明示的に生成することによって、フォーマットされた文書を作成できます。
- **HTML**　文またはプロシージャの結果セットは、テーブルを格納する HTML ドキュメントに自動的にフォーマットされます。
- **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 以外の場合は、自動的に XML としてフォーマットされます。効果は、SELECT 文で FOR XML RAW 句を使用した場合と同様です。
- **SOAP**　結果セットは、SOAP 応答として返されます。データのフォーマットは、FORMAT 句によって決定されます。SOAP サービスへの要求は、単純な HTTP 要求ではなく有効な SOAP 要求である必要があります。SOAP 規格の詳細については、www.w3.org/TR/SOAP を参照してください。
- **DISH**　DISH サービス (SOAP ハンドラを決定) は、GROUP 句で識別される SOAP サービスのプロキシとして機能し、これらの各 SOAP サービスに対して WSDL (Web Services Description Language) ファイルを生成します。

備考

ALTER SERVICE 文は、ISYSWEBSERVICE システム・テーブルを変更するときに使用します。結果として、データベース・サーバは Web サーバとして動作することができます。

パーミッション

DBA 権限が必要です。

関連する動作

なし。

参照

- 「SOAP サービスの使用」 『SQL Anywhere サーバ - プログラミング』
- 「CREATE SERVICE 文」 448 ページ
- 「DROP SERVICE 文」 521 ページ
- 「SYSWEBSERVICE システム・ビュー」 837 ページ
- 「-xs サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』
- 「SQL Anywhere Web サービス」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

Web サーバをすばやく設定するには、-xs (http または https) オプションを使用してデータベース・サーバを起動し、次の文を実行します。

```
CREATE SERVICE tables TYPE 'HTML';

ALTER SERVICE tables
  AUTHORIZATION OFF
  USER DBA
  AS SELECT *
    FROM SYS.SYSTAB;
```

これらの文を実行したら、Web ブラウザを使用して URL *http://localhost/tables* を開きます。

ALTER STATISTICS 文

この構文は、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するときに使用します。

構文

ALTER STATISTICS
[ON] *table* **[(** *column1* **[,** *column2* **...** **])]**
AUTO UPDATE { ENABLE | DISABLE }

パラメータ

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・サーバはオプティマイザ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 権限が必要です。

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」 452 ページ
- 「DROP STATISTICS 文」 523 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 SUBSCRIPTION 文 [Mobile Link]

この文を使用して、SQL Anywhere リモート・データベースで、Mobile Link ユーザによるパブリケーションへの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
TO *publication-name*
[**FOR** *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

TO 句 パブリケーション名を指定します。

FOR 句 1 つ以上の Mobile Link ユーザ名を指定します。

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。

異なるロケーションで指定されるオプションを dbmlsync が処理する方法については、「優先順位」 『Mobile Link - クライアント管理』を参照してください。

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。

通信プロトコルの詳細については、「CommunicationType (ctp) 拡張オプション」 『Mobile Link - クライアント管理』を参照してください。

ADDRESS 句 Mobile Link 同期サーバ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。

プロトコル・オプションの完全なリストについては、「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』を参照してください。

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。

各オプションの値に、"=", ",", ";" の記号は使用できません。

オプションの完全なリストについては、「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』を参照してください。

備考

network-protocol、*protocol-options*、*options* は、複数の個所で設定できます。

異なるロケーションで指定されるオプションを dbmlsync が処理する方法については、「優先順位」『Mobile Link - クライアント管理』を参照してください。

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。

「dbmlsync 構文」『Mobile Link - クライアント管理』を参照してください。

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

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 436 ページ
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」 518 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 での同期の設計」『Mobile Link - クライアント管理』
- 「ISYSSYNC システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、Mobile Link サーバのアドレスを変更します。

```
ALTER SYNCHRONIZATION SUBSCRIPTION
TO p1
FOR ml1
TYPE TCPIP
ADDRESS 'host=10.11.12.132;port=2439';
```


ALTER SYNCHRONIZATION USER 文 [Mobile Link]

この文を使用して、SQL Anywhere リモート・データベースで Mobile Link ユーザ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

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。

通信プロトコルの詳細については、「CommunicationType (ctp) 拡張オプション」『Mobile Link - クライアント管理』を参照してください。

ADDRESS 句　Mobile Link サーバ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。

プロトコル・オプションの完全なリストについては、「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』を参照してください。

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。

オプションの完全なリストについては、「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』を参照してください。

備考

network-protocol、*protocol-options*、*options* は、複数の個所で設定できます。

異なるロケーションで指定されるオプションを dbmlsync が処理する方法については、「優先順位」『Mobile Link - クライアント管理』を参照してください。

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。

「dbmlsync 構文」 『Mobile Link - クライアント管理』を参照してください。

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

- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」 458 ページ
- 「DROP SYNCHRONIZATION USER 文 [Mobile Link]」 527 ページ
- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「ISYSSYNC システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

ALTER TABLE 文

この文は、テーブル定義を修正するとき、従属ビューを無効にするとき、またはテーブルの Replication Server レプリケーションを有効にするときに使用します。

構文

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*
| **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
| **REPLICATE** { **ON** | **OFF** }
| [**NOT**] **ENCRYPTED**

new-column-attribute :
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* **)**
| *built-in-function* **(** *constant-expression* **)**
| **AUTOINCREMENT**
| **GLOBAL AUTOINCREMENT** [**(** *partition-size* **)**]
| **NULL**
| **TIMESTAMP**
| **UTC TIMESTAMP**
| **LAST USER**
| **USER**

special-value :
CURRENT {
  DATABASE
  | **DATE**
  | **REMOTE USER**
  | **TIME**
  | **TIMESTAMP**
  | **UTC TIMESTAMP**
  | **USER**
  | **PUBLISHER** }

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

ADD column-name [AS] column-data-type [new-column-attribute ...] 句　この構文は、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。指定するデータ型の詳細については、「SQL データ型」 47 ページを参照してください。

使用できるカラムの属性を次に示します。

- **[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。Bit 型のカラムでは、新規のカラムに NULL 値を使用できます。Bit 型のカラムの作成時には、NOT NULL の制約が自動的に設定されます。
- **DEFAULT 句**　カラムのデフォルト値を設定します。カラムのすべてのローはこの値で作成されます。使用できるデフォルト値については、「CREATE TABLE 文」 460 ページを参照してください。
- **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**　この句はカラムに CHECK 条件を追加するときに使用します。
  - **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。クラスタード・インデックスの詳細については、「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- **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。クラスタード・インデックスの詳細については、「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- **[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**　BLOB を格納しているとき (文字データ型とバイナリ・データ型のみ)、INLINE 句と PREFIX 句を使用して、ロー内に保存する BLOB のサイズ (バイト単位) を指定します。詳細については、「CREATE TABLE 文」 460 ページの INLINE 句と PREFIX 句の説明を参照してください。
- **[NO] INDEX**　BLOB を格納しているとき (文字データ型とバイナリ・データ型のみ)、この句を使用して、BLOB 値のインデックスを構築するかどうかを指定します。[NO] INDEX 句の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

> **注意**
> BLOB インデックスはデータベースのインデックスとは異なります。BLOB インデックスを作成すると、BLOB データへのランダム・アクセスが高速になります。一方、データベースのインデックスを作成すると、1 つ以上のカラムの値にインデックスが作成されます。

- **IDENTITY**　この句は AUTOINCREMENT と同等です。T-SQL との互換性のために用意されている句です。「CREATE TABLE 文」 460 ページの AUTOINCREMENT の説明を参照してください。
- **COMPUTE**　この句は、カラムの値に *expression* の値を反映させるときに使用します。COMPUTE 句の使用の制限については、「CREATE TABLE 文」 460 ページを参照してください。

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。

- **CHECK**　この句はテーブルに CHECK 条件を追加するときに使用します。テーブル検査制約は FALSE が返された場合にエラーとなります。UNKNOWN 値を使用して変更を有効にすることができます。この制約の詳細については、「CREATE TABLE 文」 460 ページを参照してください。
- **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。この制約の詳細については、「CREATE TABLE 文」 460 ページを参照してください。
- **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。この制約の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

クラスタード・インデックスの詳細については、「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- **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。この制約の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

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。

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、制約またはキーを削除してからカラムを修正してください。変更できる内容を次に示します。属性の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

- **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。
- **[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。
- **INLINE 句と PREFIX 句** BLOB を含むカラムで INLINE 句と PREFIX 句を使用すると、ロー内に保持する BLOB のサイズ (バイト単位) を指定できます。INLINE 句と PREFIX 句を設定す

る方法の詳細については、「CREATE TABLE 文」 460 ページの INLINE 句と PREFIX 句の説明を参照してください。

- **[NO] INDEX 句**　この句は、そのカラムのサイズが大きな BLOB にインデックスを構築するかどうかを指定するときに使用します。この句の使用方法については、「CREATE TABLE 文」 460 ページの [NO] INDEX 句に関する項を参照してください。
- **SET COMPUTE 句**　この句は、計算済みカラムと関連付けられた式を変更するときに使用します。この文を実行すると、カラムの値が再計算されます。新しい式が無効な場合、この文は失敗します。COMPUTE 式の使用の制限については、「CREATE TABLE 文」 460 ページを参照してください。

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。

DROP 句　この句は、カラム、テーブルの制約、またはインデックスを削除するときに使用します。テーブルまたはカラムで削除されるオブジェクトを次に示します。

- **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。

RENAME 句　この句は、テーブル、カラム、または制約の名前を変更するときに使用します。

- **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ることに注意してください。名前の変更操作が正常に実行された後は、ON UPDATE アクションまたは ON DELETE

アクションが指定された外部キーを削除してから再作成する必要があります。これは、アクションの実装に使用された system-created トリガが古い名前を参照しないようにするためです。

- **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ることに注意してください。名前の変更操作が正常に実行された後は、ON UPDATE アクションまたは ON DELETE アクションが指定された外部キーを削除してから再作成する必要があります。これは、アクションの実装に使用された system-created トリガが古い名前を参照しないようにするためです。
- **RENAME CONSTRAINT constraint-name TO new-constraint-name**　制約の名前を *new-constraint-name* に変更します。

table-alteration 句　この句は、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。
- **REPLICATE { ON | OFF }**　この句は、レプリケーション時にテーブルを含めるかどうかの設定を変更するときに使用します。テーブルに REPLICATE ON があるとき、テーブルに対するすべての変更はレプリケーションのために Replication Server に送信されます。Replication Server のレプリケーション定義は、どのテーブルの変更が他のサイトに送信されたかを決定するために使用されます。
- **[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、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テーブルが暗号化されます。「テーブル暗号化を有効にする」 『SQL Anywhere サーバ - データベース管理』を参照してください。テーブルを暗号化した後でも、テンポラリ・ファイルとトランザクション・ログに含まれるテーブルのデータは暗号化されない形式で保存されます。テンポラリ・ファイルを削除するにはデータベースを再起動します。-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。詳細については、「バックアップ・ユーティリティ (dbbackup)」 『SQL Anywhere サーバ - データベース管理』または「BACKUP 文」 350 ページを参照してください。

  テーブルの暗号化が有効な場合、暗号化された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・ファイルのページが暗号化されます。さらに、暗号化された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・ページも暗号化されます。

DISABLE VIEW DEPENDENCIES 句　この句は、テーブルに依存している非実体化ビュー (Non-materialized View) を無効にするときに使用します。この句で無効にしたビューを改めて有効にするには、各ビューで ALTER VIEW ... ENABLE 文を実行します。この句では、依存している実体化ビュー (Materialized View) は無効になりません。無効にするには、ALTER MATERIALIZED VIEW ... DISABLE 文を使用します。このとき、依存している実体化ビュー (Materialized View) がある場合でも、この句をテーブルに使用することはできません。「ALTER MATERIALIZED VIEW 文」 316 ページを参照してください。

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。テーブルのスキーマを変更すると、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実体化ビュー (Materialized View) 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く必要があります。ビューの依存関係については、「ビューの依存性」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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 は、文が他の接続で現在使用中のテーブルに影響するときには防止されます。ALTER TABLE には時間がかかり、データベース・サーバは、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。

CLUSTERED オプションの使い方の詳細については、「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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
- 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

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。「スナップショット・アイソレーション」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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、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。新たに統計情報を生成する方法については、「カラム統計の更新」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

参照

- 「CREATE TABLE 文」 460 ページ
- 「DROP 文」 512 ページ

- 「SQL データ型」 47 ページ
- 「テーブルの変更」 『SQL Anywhere サーバ - SQL の使用法』
- 「特別値」 30 ページ
- 「テーブル制約とカラム制約の使い方」 『SQL Anywhere サーバ - SQL の使用法』
- 「allow_nulls_by_default オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「テーブル暗号化を有効にする」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ADD COLUMN はコア機能です。他の句は、ベンダ拡張または SQL/2003 への特定の名前付き拡張の実装です。

例

次の例は、従業員が勤めている事務所を示す新しいカラムを、Employees テーブルに追加します。

```
ALTER TABLE Employees
ADD Office CHAR(20) DEFAULT 'Boston';
```

次の例は、Employee テーブルから Office カラムを削除します。

```
ALTER TABLE Employees
DROP Office;
```

Customer 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、Customer 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。

ALTER TRIGGER 文

この文は、トリガの定義を修正したものに置き換える場合に使用します。

新しいトリガの定義全体を ALTER TRIGGER 文にインクルードします。

構文 1

ALTER TRIGGER *trigger-name trigger-definition*

trigger-definition : CREATE TRIGGER syntax

構文 2

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。*trigger-definition* の詳細については、「CREATE TRIGGER 文」 473 ページと「CREATE TRIGGER 文 [T-SQL]」 479 ページを参照してください。

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、関連するトリガの定義にスクランブルをかけ、解読できないようにします。このトリガはアンロードして、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、デバッガを使用したデバッグでも、プロシージャ・プロファイリングによっても、トリガ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

トリガを定義するテーブルの所有者であるか、ユーザ 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あり、さらに RESOURCE 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE TRIGGER 文」 473 ページ
- 「CREATE TRIGGER 文 [T-SQL]」 479 ページ
- 「DROP 文」 512 ページ
- 「プロシージャ、関数、トリガ、ビューの内容を隠す」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　ベンダ拡張。

ALTER VIEW 文

この文は、ビューの定義を修正したものに置き換える場合に使用します。

構文 1

ALTER VIEW
[*owner*.]*view-name* [**(** *column-name*, … **)**] **AS** *select-statement*
[**WITH CHECK OPTION**]

構文 2

ALTER VIEW
[*owner*.]*view-name* { **SET HIDDEN** | **RECOMPILE** | **DISABLE** | **ENABLE** }

パラメータ

AS 句 この句の用途と構文は、CREATE VIEW 文と同じです。「CREATE VIEW 文」 482 ページを参照してください。

WITH CHECK OPTION 句 この句の用途と構文は、CREATE VIEW 文と同じです。「CREATE VIEW 文」 482 ページを参照してください。

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。ただし、難読化しても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。

DISABLE 句 DISABLE 句は、データベース・サーバ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

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。ビューの変更とビューの依存関係に影響を及ぼす内容の詳細については、「ビューの依存性」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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。ビュー構造の定義に使用するパラメータの詳細については、「CREATE VIEW 文」 482 ページを参照してください。

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、デバッガを使用したデバッグでも、プロシージャ・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とトリガ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やトリガ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、プロシージャとトリガ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」 482 ページ
- 「DROP 文」 512 ページ
- 「プロシージャ、関数、トリガ、ビューの内容を隠す」 『SQL Anywhere サーバ - SQL の使用法』
- 「ビューの依存性」 『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE MATERIALIZED VIEW 文」 420 ページ
- 「ALTER MATERIALIZED VIEW 文」 316 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

ATTACH TRACING 文

診断トレーシング・セッションを開始するときに、この文を使用します。つまり、診断テーブルに診断情報を送信し始めるときに使用します。

構文

ATTACH TRACING TO { **LOCAL DATABASE** | *connect-string* }
[**LIMIT** { *size* | *history* }]

connect-string : the connection string for the database

size : **SIZE** *nnn* { **MB** | **GB** }

history : **HISTORY** *nnn* { **MINUTES** | **HOURS** | **DAYS** }

nnn : *integer*

パラメータ

connstr　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。

limit　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、主に Sybase Central の [診断トレーシング] ウィザードで使用します。手動で実行することもできます。データベースのプロファイルを使用する場合は、手動で実行する必要があります。

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・サーバで TCP/IP が実行されている必要があります。「TCP/IP プロトコルの使用」『SQL Anywhere サーバ - データベース管理』を参照してください。

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・インタフェー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。「sa_diagnostic_tracing_level テーブル」 775 ページを参照してください。

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。「データベース・レベルのプロパティ」『SQL Anywhere サーバ - データベース管理』を参照してください。

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 権限が必要です。

関連する動作

なし。

参照

- 「DETACH TRACING 文」 510 ページ
- 「REFRESH TRACING LEVEL 文」 642 ページ
- 「診断トレーシングを使用した詳細なアプリケーション・プロファイリング」『SQL Anywhere サーバ - SQL の使用法』
- 「sa_set_tracing_level システム・プロシージャ」 960 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、sa_set_tracing_level システム・プロシージャを使用して、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;eng=remotedbsrv10;dbn=mytracingdb;links=tcpip'
 LIMIT HISTORY 2 HOURS;
```


BACKUP 文

この文は、データベースとトランザクション・ログをバックアップするために使用します。

構文 1 (イメージのバックアップ)

BACKUP DATABASE
DIRECTORY *backup-directory*
[**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** }]

backup-directory : { *string* | *variable* }

構文 2 (アーカイブのバックアップ)

BACKUP DATABASE TO *archive-root*
[**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** }]

archive-root : { *string* | *variable* }

comment-string : *string*

パラメータ

backup-directory 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して、ログをコピーせずに名前を変更したり、トランケートすることができます。

WAIT BEFORE START 句 この句を指定すると、データベースのバックアップ・コピーにはリカバリに必要な情報は含まれません。特に、各接続のロールバック・ログが空になります。

この句を指定してバックアップを実行すると、データベースのバックアップ・コピーを読み込み専用で開始し、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。

WAIT AFTER END 句 この句は、トランザクション・ログの名前変更またはトランケートを行う場合に使用します。この句を指定すると、すべてのトランザクションが完了してから、ログの

名前変更またはトランケートが行われます。この句が使用されている場合、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ます。

DBFILE ONLY 句　この句は、メイン・データベース・ファイルと関連する DB 領域のバックアップ・コピーを作成するのに使用されます。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　この句は、トランザクション・ログのバックアップ・コピーを作成するのに使用します。他のデータベース・ファイルはコピーされません。

TRANSACTION LOG RENAME [MATCH] 句　この句を指定すると、サーバによって、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、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。

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。

archive-root　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。たとえば、NetWare では、最初のテープ・ドライブは *¥¥.¥tape0* となります。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。エスケープ文字と文字列の詳細については、「文字列」 ページを参照してください。

ON EXISTING ERROR　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* 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。次のすべての条件があてはまる場合は、ファイルが更新されないようにしたい場合があります。

- 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・ディレクトリにバックアップ・ファイルを書き込む専用の 1 スレッドが割り当てられます。ただし、バックアップ・ディレクトリが増加したライタの負荷を処理できるデバイス上にある場合 (RAID アレイなど)、ライタ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、データベース・サーバは、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検証します。追加のライタ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・サーバで追加のライタを作成します。このオプションはデフォルトで ON です。

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。デフォルト設定値は AUTO です。

- **COPY**　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、データベース・ファイルのバックアップ中に、変更されたページは適用されません。チェックポイント・ログの全体とシステム DB 領域がバックアップ・ディレクトリにコピーされます。このデータベースを次に起動すると、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中の他の接続との内部サーバ競合が減少します。ただし、チェックポイント・ログには修正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・コピーによって、バックアップの開始時よりもデータベース・ファイルのサイズが大きくなる可能性があります。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**　NO COPY を指定すると、チェックポイント・ログがバックアップ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** 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・ファイルのバックアップは、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) となります。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** AUTO を指定すると、データベース・サーバはバックアップ・ディレクトリがあるボリュームの空きディスク領域サイズをチェックします。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は、COPY を指定した場合と同様にオプションが動作します。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。

備考

BACKUP 文は、サーバ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。「バックアップ・ユーティリティ (dbbackup)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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* ファイルの場所を指定する方法については、「SALOGDIR 環境変数」 『SQL Anywhere サーバ - データベース管理』を参照してください。

構文 1 (イメージのバックアップ) イメージのバックアップでは、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。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、ログをコピーせずに名前を変更したり、トランケートすることができます。これは、スペースが問題となるレプリケーション環境では特に便利です。この機能をトランザクション・ログ・サイズのイベント・ハンドラと一緒に使用して、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、ログが必要なくなったときに削除したりすることができます。

イメージのバックアップからリストアを行うには、保存されたファイルを元のロケーションにコピーして、「バックアップとデータ・リカバリ」 『SQL Anywhere サーバ - データベース管理』の説明に従ってトランザクション・ログを再適用します。

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 文を使用してください。

RESTORE DATABASE 文が、トランザクション・ログのみ含むアーカイブ・ファイルを参照している場合、復元されたデータベース・ファイルの場所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、ログのみ含むアーカイブからディレクトリ *C:¥MYNEWDB* に復元するには、RESTORE DATABASE 文は次のようになります。

```
RESTORE DATABASE 'c:¥mynewdb¥my.db' FROM archive-root
```

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

- 「バックアップ・ユーティリティ (dbbackup)」『SQL Anywhere サーバ - データベース管理』
- 「イメージ・バックアップの作成」『SQL Anywhere サーバ - データベース管理』
- 「RESTORE DATABASE 文」 650 ページ
- 「バックアップとデータ・リカバリ」『SQL Anywhere サーバ - データベース管理』
- 「EXECUTE IMMEDIATE 文 [SP]」 536 ページ
- 「並列データベース・バックアップの知識」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。
- **Windows CE** Windows CE では、BACKUP DATABASE DIRECTORY 構文 (上記の構文 1) だけ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。イメージの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';
```

コピーを作成せずにログの名前を変更します。

```
BACKUP DATABASE DIRECTORY ' '
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

動的に構成された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 文

この文は SQL 文を 1 つにまとめるために使用します。

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*

variable-declaration :
DECLARE *variable-name data-type*

exception-declaration :
DECLARE *exception-name* **EXCEPTION**
FOR SQLSTATE [**VALUE**] *string*

exception-case :
 WHEN *exception-name* [, …] **THEN** *statement-list*
| **WHEN OTHERS THEN** *statement-list*

パラメータ

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。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には表示されません。

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。プロシージャまたはトリガの本文である複合文は、プロシージャまたはトリガの名前と同じ暗黙のラベルを持っています。

複合文と例外処理の詳細については、「プロシージャとトリガでのエラーと警告」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

ATOMIC　ATOMIC 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文です。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。

備考

プロシージャまたはトリガの本文は複合文です。複合文は、プロシージャまたはトリガ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

- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ
- 「DECLARE LOCAL TEMPORARY TABLE 文」 495 ページ
- 「CONTINUE 文 [T-SQL]」 378 ページ
- 「SIGNAL 文」 696 ページ
- 「RESIGNAL 文」 649 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「アトミックな複合文」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** 永続的ストアド・モジュール機能。

例

プロシージャまたはトリガ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:
  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;
```

BEGIN TRANSACTION 文 [T-SQL]

この文は、ユーザ定義のトランザクションを開始するために使用します。

構文

BEGIN TRAN[SACTION] [*transaction-name*]

備考

オプションのパラメータ *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。

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/2003 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だけ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。「chained オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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 chained オプションを変更することはできません。

> **警告**
> ストアド・プロシージャを呼び出すときは、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 変数は、最初の BEGIN TRANSACTION 文が実行される前に 0 の値を持ち、@@trancount が 1 のときに実行した 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。

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」 372 ページ
- 「isolation_level オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「ROLLBACK 文」 663 ページ
- 「SAVEPOINT 文」 668 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 - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次のバッチは、@@trancount の連続値を 0、1、2、1、0 で報告します。値は [サーバ・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
```

@@trancount の値は、発行された明示的な 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。その結果、BEGIN TRANSACTION 文が始まる前 (現在のトランザクションがあるときでも) は、SQL Anywhere の @@trancount 変数は 0 の値を持ち、Adaptive Server Enterprise (連鎖モードで) 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。

BREAK 文 [T-SQL]

この文は、複合文またはループから出るために使用します。

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。

関連する動作

なし。

参照

- 「WHILE 文 [T-SQL]」 744 ページ
- 「CONTINUE 文 [T-SQL]」 378 ページ
- 「BEGIN 文」 356 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** 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

この文は、プロシージャを呼び出すために使用します。

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を呼び出します。プロシージャが完了すると、INOUT または OUT パラメータ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。順序を守って指定する場合、引数はプロシージャのパラメータ・リスト内の対応するパラメータと一致します。キーワードを使用した場合、引数は指定したパラメータと一致します。

プロシージャ引数には、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。

プロシージャの内部では、プロシージャが結果セットを返す場合、DECLARE 文で CALL 文を使用できます。「プロシージャから返される結果」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

プロシージャは RETURN 文を使って整数値 (ステータス・インジケータとして) 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, ... )
```

パーミッション

プロシージャの所有者であるか、そのプロシージャに対する 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」 408 ページ
- 「CREATE PROCEDURE 文」 423 ページ
- 「GRANT 文」 565 ページ
- 「EXECUTE 文 [T-SQL]」 534 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

例

ShowCustomers プロシージャを呼び出します。このプロシージャはパラメータがなく、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を作成し、結果を保持する変数を作成し、プロシージャを呼び出し、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

この文は、複数の状況に基づいた実行パスを選択するために使用します。

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 が null でよい場合は、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　このフォームの文は、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
```

CASE 文は CASE 式とは異なります。
CASE 文の構文と CASE 式の構文を混同しないでください。「CASE 式」 17 ページを参照してください。

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]」 187 ページ

- 「不定の値：NULL」 『SQL Anywhere サーバ - SQL の使用法』
- 「BEGIN 文」 356 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** 永続的ストアド・モジュール機能。

例

CASE 文を使用する次のプロシージャは、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;
  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
```

CHECKPOINT 文

この文は、データベースにチェックポイントを実行させるために使用します。

構文

CHECKPOINT

備考

CHECKPOINT 文はデータベース・サーバ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 - データベース管理』
- 「checkpoint_time オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「recovery_time オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

CLEAR 文 [Interactive SQL]

この文は、Interactive SQL のウィンドウ枠をクリアするために使用します。

構文

CLEAR

備考

CLEAR 文を使用して、[SQL 文] ウィンドウ枠、[メッセージ] ウィンドウ枠と、[結果] ウィンドウ枠の [結果]、[メッセージ]、[プラン] の各タブをクリアします。

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。

標準と互換性

- **SQL/2003**　ベンダ拡張。

CLOSE 文 [ESQL] [SP]

この文は、カーソルを閉じるために使用します。

構文

CLOSE *cursor-name*

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]」 620 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ
- 「PREPARE 文 [ESQL]」 629 ページ

標準と互換性

- **SQL/2003** コア機能。

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

この文は、データベース・オブジェクトに対するシステム・テーブルにコメントを格納するために使用します。

構文

COMMENT ON
{
  COLUMN [*owner*.]*table-name.column-name*
 | **EVENT** *event-name*
 | **FOREIGN KEY** [*owner*.]*table-name.role-name*
 | **INDEX** [[*owner*.] *table*.]*index-name*
 | **JAVA CLASS** *java-class-name*
 | **JAVA JAR** *java-jar-name*
 | **INTEGRATED LOGIN** *integrated-login-id*
 | **PROCEDURE** [*owner*.]*procedure-name*
 | **SERVICE** *web-service-name*
 | **TABLE** [*owner*.]*table-name*
 | **TRIGGER** [[*owner*.]*tablename*.]*trigger-name*
 | **USER** *userid*
 | **VIEW** [*owner*.]*view-name*
 | **MATERIALIZED VIEW** [*owner*.]*materialized-view-name*
 | **PRIMARY KEY ON** [*owner*.]*table-name*
 | **KERBEROS LOGIN "***client-Kerberos-principal***"**
}
IS *comment*

comment : *string* | **NULL**

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。

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/2003**　ベンダ拡張。

例

次の例は、コメントの追加と削除の方法を示します。

Employee テーブルにコメントを追加します。

```
COMMENT
ON TABLE Employees
IS 'Employee information';
```

Employee 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';
```

COMMIT 文

この文を使用して、データベースを永続的に変更したり、ユーザ定義のトランザクションを終了します。

構文 1

COMMIT [WORK]

構文 2

COMMIT TRAN[SACTION] [*transaction-name*]

パラメータ

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]」 359 ページを参照してください。セーブポイントの詳細については、「SAVEPOINT 文」 668 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 [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「cooperative_commits オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「delayed_commits オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「delayed_commit_timeout オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』

コミット接続プロパティを使用して、現在の接続のコミット数を取得できます。「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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継続保管します。

データ定義文はすべて、コミットを自動的に実行します。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・サーバが無効な外部キーを検知すると、COMMIT 文は失敗します。その場合、無効な外部キーがあることによってトランザクションが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。

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。

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。

参照

- 「SAVEPOINT 文」 668 ページ
- 「BEGIN TRANSACTION 文 [T-SQL]」 359 ページ
- 「PREPARE TO COMMIT 文」 631 ページ
- 「ROLLBACK 文」 663 ページ

標準と互換性

- **SQL/2003**　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]

この文は、Interactive SQL の [オプション] ダイアログを開くために使用します。

構文

CONFIGURE

備考

CONFIGURE 文は、Interactive SQL の [オプション] ダイアログを開きます。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。このダイアログで 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

- 「SET OPTION 文」 686 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

CONNECT 文 [ESQL] [Interactive SQL]

この文は、データベースへの接続を確立するために使用します。

構文 1

CONNECT
[**TO** *database-server-name*]
[**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

CONNECT USING *connect-string*

connect-string : { *identifier* | *string* | *hostvar* }

パラメータ

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 *connect-string* はセミコロンで区切った keyword=*value* 形式のパラメータ設定リストです。一重引用符で囲んでください。

接続文字列の詳細については、「接続パラメータ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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、トリガ、イベント、またはバッチではサポートされていません。

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 (最初に起動するデータベース・サーバ) が使用されます。*database-name* を指定しない場合、指定されたサーバ上の最初のデータベースが使用されます。

WHENEVER、SET SQLCA、いくつかの 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 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。

接続アルゴリズムの詳細については、「接続のトラブルシューティング」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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を指定しない場合、Interactive SQL はデフォルトのサーバ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名を指定する場合、Interactive SQL は指定したサーバのデフォルト・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・プロンプトから 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 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。「SETUSER 文」 694 ページを参照してください。

Interactive SQL では、接続情報 (データベース名、ユーザ ID、データベース・サーバなど) 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 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。

参照

- 「GRANT 文」 565 ページ
- 「DISCONNECT 文 [ESQL] [Interactive SQL]」 511 ページ
- 「SET CONNECTION statement [Interactive SQL] [ESQL]」 683 ページ
- 「SETUSER 文」 694 ページ
- 「接続パラメータ」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** 構文 1 はコア SQL に含まれない SQL／基本機能です。構文 2 はベンダ拡張です。

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 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 demo10
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 ID、データベース・サーバ名が 「**demo10** 上の **demo** (DBA)」 と表示されます。

CONTINUE 文 [T-SQL]

この文は、ループを再開するために使用します。

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。Watcom-SQL を使用して複数の文内に CONTINUE を使用するには、*statement-label* を指定する必要があります。

Transact-SQL を使用して複数の文内に CONTINUE を使用するには、*statement-label* は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」 614 ページ
- 「WHILE 文 [T-SQL]」 744 ページ
- 「FOR 文」 547 ページ
- 「BEGIN 文」 356 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

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 ー 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;
  END LOOP lbl;
END
```

CREATE DATABASE 文

この文は、データベースを作成するために使用します。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。

構文

CREATE DATABASE *db-file-name-string*
[**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-file-name-string*]
[**MIRROR** *mirror-file-name-string*]] }

page-size :
2048 | **4096** | **8192** | **16384** | **32768**

algorithm-key-spec:
ON
| [**ON**] **KEY** *key* [**ALGORITHM** { **'AES'** | **'AES_FIPS'** }]
| [**ON**] **ALGORITHM** { **'AES'** | **'AES_FIPS'** } **KEY** *key*
| [**ON**] **ALGORITHM 'SIMPLE'**

パラメータ

ファイル名 (*db-file-name-string*、*log-file-name-string*、*mirror-file-name-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・サーバの作業ディレクトリを基準に作成されます。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。

- ファイル拡張子を指定しない場合、データベース・ファイルは *.db* 拡張子、トランザクション・ログには *.log* 拡張子、ミラー・ログには *.mlg* 拡張子を付けて作成されます。

ACCENT 句 ACCENT 句は、COLLATION 句または NCHAR COLLATION 句で指定した照合に UCA (Unicode Collation Algorithm) 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、文字のアクセントを考慮して UCA 文字列が比較されます。たとえば、e は e 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、アクセントを無視して文字列が比較されます。たとえば、e と e は同等です。これがデフォルトの動作です。

詳細については、「国際言語と文字セット」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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、Sybase Adaptive Server Enterprise の互換性ビュー dbo.syscolumns と dbo.sysindexes と矛盾します。

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。したがって、値 'a ' (文字 'a' と、後続の 1 つのブランク) は、単一文字列 'a' と等しくありません。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。

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。「ansi_blanks オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

CASE 句 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、b 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。

デフォルトでは、比較時に大文字と小文字が区別されません。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。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。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' );
```

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。デフォルトでチェックサムはオフです。そのため、CHECKSUM 句を省略すると OFF が適用されます。

この句の設定に関係なく、データベース・サーバは常に重要なページのチェックサムを計算します。

「検証ユーティリティ (dbvalid)」『SQL Anywhere サーバ - データベース管理』、「sa_validate システム・プロシージャ」 970 ページ、または 「VALIDATE 文」 739 ページを参照してください。

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 を使用する照合リストから選択するか、Unicode Collation Algorithm (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。「照合の選択」『SQL Anywhere サーバ - データベース管理』を参照してください。

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。大文字小文字と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ともに、ACCENT 句または CASE 句を指定すると、ACCENT 句と CASE 句の値はデフォルトとしてのみ使用されます。次の表は、サポートされているキーワードを示したものです。使用できる代替形式と値も示します。

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。

注意
UCA 照合を指定すると、以下に示す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。
また、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・サーバでは起動できません。

照合の適合化オプション

キーワード	照合	代替形式	指定可能な値
Locale	UCA	(なし)	任意の有効なロケール・コードです。たとえば、en があります。
CaseSensitivity	サポートされているすべての照合	CaseSensitive、Case	◆ **respect** 文字の大文字小文字の違いが考慮されます。UCA 照合の場合、UpperFirst を指定したことに等しくなります。その他の照合の場合は、照合によって異なります。 ◆ **ignore** 文字の大文字小文字の違いは無視されます。 ◆ **UpperFirst** 常に、大文字を先にソートします (Aa)。 ◆ **LowerFirst** 常に、小文字を先にソートします (aA)。
AccentSensitivity	UCA	AccentSensitive、Accent	◆ **respect** 文字のアクセントの違いが考慮されます。 ◆ **ignore** 文字のアクセントの違いは無視されます。 ◆ **French** フランス語の規則に従ってアクセントの違いが考慮されます。

キーワード	照合	代替形式	指定可能な値
PunctuationSensitivity	UCA	PunctuationSensitive、Punct	♦ **ignore** 句読表記の違いは無視されます。 ♦ **primary** 第 1 レベルのソートを使用します (文字のみを考慮します)。たとえば、a > b になります。 ♦ **quaternary** 第 4 レベルのソートを使用します。文字、大文字小文字、アクセント、句読表記の順に考慮されます。たとえば、multiByte、multibyte、multi-byte、multi-Byte をソートすると、次のようになります。 ♦ multiByte ♦ multibyte ♦ multi-Byte ♦ multi-byte 大文字と小文字やアクセント記号を区別しないデータベースでは、quaternary (第 4 レベル) は指定できません。
SortType	UCA	(なし)	使用するソート・タイプです。可能な値は、次のとおりです。 ♦ phonebook ♦ traditional ♦ standard ♦ pinyin ♦ stroke ♦ direct ♦ posix ♦ big5han ♦ gb2312han これらのソート・タイプの詳細については、Unicode Technical Standard #35 (http://www.unicode.org/reports/tr35/) を参照してください。

DATABASE SIZE 句　データベースが使用する領域の事前割り付けを行うと、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。デフォルトの*サイズ*はバイト単位です (BYTES で指定することもできます)。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ KB、MB、または GB を使用します。P を使用すると、値は使用できる領域の総量に対する割合を表します。PAGES を使用して、サイズをページ数で指定することもできます。

DBA USER 句　この句は、データベースの DBA ユーザ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 ID が作成されます。

DBA PASSWORD 句　DBA データベース・ユーザ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 ユーザに使用されます。

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 Collation Algorithm) 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。

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。推奨されるエンコードと照合の詳細については、「推奨文字セットと照合」 『SQL Anywhere サーバ - データベース管理』を参照してください。

SQL Anywhere でサポートされるエンコードのリストを取得する方法については、「サポートされている文字セット」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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。「テーブルの暗号化」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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はデータを解読できます。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。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。強力な暗号化では、ALGORITHM 句を使用して AES 128 ビット・アルゴリズム (AES または AES_FIPS) を指定し、KEY 句を使用して暗号化キーを指定します。

Windows CE では、ARM プロセッサ用に AES_FIPS アルゴリズムがサポートされています。

警告
キーは保護してください。キーのコピーは安全な場所に保管してください。キーを紛失すると、データベースにまったくアクセスできなくなり、リカバリも不可能になります。

強力なデータベース暗号化の詳細については、「強力な暗号化」 『SQL Anywhere サーバ - データベース管理』を参照してください。

JCONNECT 句 Sybase jConnect JDBC ドライバからシステム・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。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。

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 Collation Algorithm (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。「照合の選択」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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。大文字小文字と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ともに、ACCENT 句または CASE 句を指定すると、ACCENT 句と CASE 句の値はデフォルトとしてのみ使用されます。これらのオプションの組み合わせを指定する構文は、上記の COLLATION 句を定義する構文と同じです。「照合の適合化オプション」 382 ページを参照してください。

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。

注意
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・サーバでは起動できません。

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;
```

ページ・サイズ制限
現在のサーバ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・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・コマンド・ラインで -gp オプションを使用して設定します。

TRANSACTION LOG 句　トランザクション・ログは、データベース・サーバ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 (「トランザクション・ログ」 『SQL Anywhere サーバ - データベース管理』を参照)、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 はミラー化されたトランザクション・ログを使用しません。

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。この文は、プロシージャ、トリガ、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

この文を実行するのに必要なパーミッションは、サーバ・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。

データベース・サーバ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」 303 ページ
- 「DROP 文」 512 ページ
- 「初期化ユーティリティ (dbinit)」 『SQL Anywhere サーバ - データベース管理』
- 「DatabaseKey 接続パラメータ [DBKEY]」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 2048
```

```
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

この文は、新しい DB 領域を定義し、対応するデータベース・ファイルを作成するために使用します。

構文

CREATE DBSPACE *dbspace-name* **AS** *file-name*

パラメータ

dbspace-name　データベース・ファイルの内部名です。*file-name* パラメータはデータベース・ファイルの実際の名前であり、必要に応じてパスを付けます。SYSTEM、TEMPORARY、TEMP、TRANSLOG、TRANSLOGMIRROR は事前定義の DB 領域に使用されているため、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。「事前定義の DB 領域」『SQL Anywhere サーバ - データベース管理』を参照してください。

file-name　明示的なディレクトリ指定がない *file-name* は、メイン・データベース・ファイルと同じディレクトリ内で作成されます。すべての相対ディレクトリはメイン・データベース・ファイルを基準にします。*file-name* にディレクトリ指定を含める場合、データベース・サーバを基準にします。NetWare 対応のデータベース・サーバを使用していて、*file-name* で絶対ディレクトリを指定する場合はボリューム名 (ドライブ文字ではなく) を使用してください。

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。

それぞれのテーブルは、1 つのデータベース・ファイル内に全体が含まれています。CREATE TABLE 文の IN 句は、テーブルを入れる DB 領域を指定します。デフォルトでは、テーブルはメイン・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 権限が必要です。

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

- 「default_dbspace オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「DROP 文」 512 ページ
- 「追加 DB 領域の使用」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

DB 領域として library を作成し、LibraryBooks テーブルとそのインデックスを保持します。

```
CREATE DBSPACE library
AS 'c:¥¥library.db';
CREATE TABLE LibraryBooks (
  title char(100),
  author char(50),
  isbn char(30),
) IN library;
```

CREATE DECRYPTED FILE 文

この文は、強力に暗号化されているデータベースを復号化します。

構文

CREATE DECRYPTED FILE *newfile*
FROM *oldfile* **KEY** *key*

パラメータ

FROM　暗号化ファイル名の一覧が表示されます。

KEY　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、暗号化されたデータベース、トランザクション・ログ・ファイル、または DB 領域を復号化し、暗号化されていないファイルを新しく作成します。オリジナル・ファイルは、暗号化キーを使用して強力に暗号化してください。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・ファイルも復号化します。また、新しいデータベースのリカバリも必要になります。

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。

既存のデータベースを暗号化する場合は、CREATE ENCRYPTED FILE 文を使用するか、または dbunload -an オプションと一緒に -ek または -ep を使用してデータベースをアンロードし、再ロードする必要があります。また、この方法を使用して、既存の暗号化キーを変更することもできます。

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。テーブルを復号する場合、ALTER TABLE 文の NOT ENCRYPTED 句を使用して復号します。「ALTER TABLE 文」 336 ページを参照してください。

この文は、プロシージャ、トリガ、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 権限が必要です。

関連する動作

なし。

参照

- 「CREATE ENCRYPTED FILE 文」 394 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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

この文は、データベース内にドメインを作成するために使用します。

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

DOMAIN | DATATYPE　CREATE DOMAIN は ANSI/ISO SQL3 の用語なので、CREATE DATATYPE ではなく、CREATE DOMAIN を使用することをおすすめします。

NULL
この句を使用すると、ドメインの Null 入力可能を指定できます。ドメインを使用してカラムを定義する場合、Null 入力可能は次のように決まります。

- 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 入力可能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　CHECK 条件を作成する場合は、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。こうすると CHECK 条件をデータ型上で定義でき、どのような名前のカラムでもこれを使用できます。

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、データ型上で定義するカラムによって継承されます。カラム上で明示的に指定された条件または値は、データ型に指定された条件を上書きします。

データベースからデータ型を削除するには DROP 文を使います。ドメインを削除するには、データ型の所有者であるか、DBA 権限が必要です。

パーミッション

RESOURCE 権限が必要です。

関連する動作

オートコミット。

参照

- 「DROP 文」 512 ページ
- 「SQL データ型」 47 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL／基本機能。

例

次の文は、35 文字の文字列を保持し、NULL が使用できる address という名前のデータ型を作成します。

```
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 という名前のデータ型を作成します。

```
CREATE DOMAIN ID INT
NOT NULL
DEFAULT AUTOINCREMENT;
```

CREATE ENCRYPTED FILE 文

この文は、暗号化されていないデータベース、トランザクション・ログ・ファイル、または DB 領域を暗号化します。暗号化されたデータベース、またはテーブルの暗号化が有効なデータベースの暗号化キーを変更するときにも使用できます。

構文

CREATE ENCRYPTED FILE *newfile*
FROM *oldfile*
{ **KEY** *key* | **KEY** *key* **OLD KEY** *oldkey* }
[**ALGORITHM** { **'AES'** | **'AES_FIPS'** }]

パラメータ

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、デフォルトとして AES が使用されます。

備考

CREATE ENCRYPTED FILE 文は次の場合に使用します。

- 暗号化されていないデータベース、トランザクション・ログ、または DB 領域を使用して、指定したキーで暗号化した新規ファイルを作成する場合。
- 暗号化されたデータベース、トランザクション・ログ、または DB 領域を使用して、新しい暗号化キーで暗号化した新規ファイルを作成する場合。

CREATE ENCRYPTED FILE 文は、新規ファイル (*newfile*) を作成しますが、前バージョンのファイル (*oldfile*) は置換または削除されません。

この文を使用してデータベースを暗号化する場合は、同じ暗号化キーを使用して対応するトランザクション・ログ・ファイル (と DB 領域) も暗号化しなければ、データベースを使用できません。暗号化されたファイルと暗号化されていないファイルを混在させることはできません。また、暗号化されたファイルに異なる暗号化アルゴリズムやキーを混在させることもできません。

リカバリを必要とするデータベースを暗号化する場合は、対応するトランザクション・ログ・ファイルも暗号化します。また、新しいデータベースのリカバリも必要になります。

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。

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ことはできます。

この文は、プロシージャ、トリガ、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 権限が必要です。

Windows CE では、ARM プロセッサ用に FIPS アルゴリズムがサポートされています。

関連する動作

なし。

参照

- 「データベースの暗号化」『SQL Anywhere サーバ - データベース管理』
- 「CREATE DECRYPTED FILE 文」 390 ページ
- 「アンロード・ユーティリティ (dbunload)」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

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 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。ログの名前が変更されてもデータベース・ファイルは引き続き古いログを指しているため、dblog −ek abcd −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 'Te9g7765*Noo';
```

次の例は、contacts データベースと contacts ログ・ファイルを暗号化して、元のログ・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';
```

次の例は、contacts データベースの暗号化キーを変更します。


```
CREATE ENCRYPTED FILE 'newcontacts.db'
 FROM 'contacts.db'
 KEY 'newkey' OLD KEY 'oldkey';
 DEL contacts.db
 RENAME newcontacts.db contacts.db;
```

CREATE EVENT 文

この文は、イベントまたは、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・ハンドラを定義するために使用します。また、スケジュールされたアクションを定義するときにも使用します。

構文

CREATE EVENT *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** | **"Disconnect"**
| **GlobalAutoincrement** | **GrowDB**
| **GrowLog** | **GrowTemp**
| **LogDiskSpace** | **MirrorFailover**
| **MirrorServerDisconnect** | **"RAISERROR"**
| **ServerIdle** | **TempDiskSpace**

trigger-condition :
event_condition(*condition-name* **)** { **=** | **<** | **>** | **!=** | **<=** | **>=** } *value*

schedule-spec :
[*schedule-name*]
　{ **START TIME** *start-time* | **BETWEEN** *start-time* **AND** *end-time* }
　[**EVERY** *period* { **HOURS** | **MINUTES** | **SECONDS** }]
　[**ON** { **(** *day-of-week*, … **)** | **(** *day-of-month*, … **)** }]
　[**START DATE** *start-date*]

event-name | *schedule-name* : *identifier*

day-of-week : *string*

day-of-month | *value* | *period* : *integer*

start-time | *end-time* : *time*

start-date : *date*

パラメータ

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であり、イベント・ハンドラ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・プロシージャの実行と同じです。他のユーザ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する条件を指定するには、WHERE 句を使用します。以下にリストされていないイベントタイプの説明については、「システム・イベントの概要」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **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・ハンドラがある場合、データベース・サーバは、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・ログとミラーリング・トランザクション・ログ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 CE 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を通知します。

  データベース・サーバの起動時に -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。

  「-fc サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **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。このハンドラの一般的なアクションは、このイベントのパラメータ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・ハンドラがある場合、データベース・サーバは 30 秒おきにサーバ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・サーバからミラー・サーバまたは監視サーバへの接続が失われたときに発生します。MirrorFailover イベントは、サーバがデータベースの所有権を取得したときに発生します。「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 - データベース管理』を参照してください。

WHERE 句　トリガ条件は、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。

有効な引数の詳細については、「EVENT_CONDITION 関数 [システム]」 159 ページを参照してください。

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・ハンドラに定義された関連するアクションのトリガ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、スケジュール名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、スケジュール名は省略できます。

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されたイベントを作成する場合に、そのイベント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**　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 DATE を指定した場合、START TIME はその日付を参照します。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。
- **BETWEEN … AND**　1 日のうち、スケジュールされた時刻が発生する範囲。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**　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 で指定された範囲内でのみ発生します。

- **ON**　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、[サーバ・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 ー 31 の整数で指定します。0 は月の末日を表します。

- **START DATE**　スケジュールされたイベントが開始される日付。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・ハンドラが完了するたびに、次のスケジュールされた時刻と日付が計算されます。

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 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。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ます。
3. 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　イベント・ハンドラ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条件が発生しても、イベント・ハンドラは起動しません。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イベントを実行する場合は、この句を使用して、イベントを処理するデータベースを制限できます。デフォルトでは、すべてのデータベースがイベントを実行します。

HANDLER 句　各イベントが 1 つのハンドラを持ちます。

備考

イベントは次の 2 つの方法で使用できます。

- **アクションのスケジュール**　データベース・サーバが一連のアクションをスケジュールに従って実行します。この機能を利用すると、バックアップ、妥当性検査、レポート・テーブル作成用のクエリなどをスケジュールできます。
- **イベント処理アクション**　データベース・サーバは、事前に定義されたイベントが発生したときに一連のアクションを実行します。処理できるイベントには、ディスク領域の制限 (ディスク領域の使用量が指定の割合を超えたとき)、データベース・サーバがアイドル状態のときなどがあります。イベント・ハンドラの動作は、実行中にエラーが検出されなければコミットされ、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条件の役割を果たします。トリガ条件が満たされると、イベント・ハンドラが実行されます。イベント・ハンドラには、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します。開発段階では TRIGGER EVENT を使ってイベント・ハンドラをテストし、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。

イベント・エラーはデータベース・サーバ・コンソールにログされます。

イベント・ハンドラ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・ハンドラがトリガされると、データベース・サーバは event_parameter 関数を使用して、イベントをトリガした接続 ID などのコンテキスト情報をイベント・ハンドラに渡します。event_parameter の詳細については、「EVENT_PARAMETER 関数 [システム]」 161 ページを参照してください。

パーミッション

DBA 権限が必要です。

イベント・ハンドラは、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 以外のパーミッションで実行するには、イベント・ハンドラからプロシージャを呼び出します。プロシージャは、その所有者の権限を使用して実行されます。イベント・ハンドラが実行される接続は、パーソナル・データベース・サーバ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」 356 ページ
- 「ALTER EVENT 文」 311 ページ
- 「COMMENT 文」 370 ページ
- 「DROP 文」 512 ページ
- 「TRIGGER EVENT 文」 717 ページ
- 「EVENT_PARAMETER 関数 [システム]」 161 ページ
- 「システム・イベントの概要」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

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;
```

「イベントのトリガ条件の定義」 『SQL Anywhere サーバ - データベース管理』を参照してください。

CREATE EXISTING TABLE 文

この文は、リモート・サーバ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*

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

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。たとえば、次の文は、テーブル a1 を MS Access ファイル *mydbfile.mdb* にマップします。

```
CREATE EXISTING TABLE a1
AT 'access;d:¥mydbfile.mdb;;a1';
```

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・オブジェクトのどちらか) 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・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。これにより、サーバ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。

カラム定義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。カラム定義が指定されている場合、SQL Anywhere

は、そのカラム定義を検証します。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。別のユーザのテーブルを作成するには、DBA 権限が必要です。

Windows CE 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」 460 ページ
- 「プロキシ・テーブルのロケーションの指定」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

リモート・サーバ 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 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 demo10 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 'demo10...Employees';
```


CREATE EXTERNLOGIN 文

この文は、リモート・サーバ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るために使用します。

構文

CREATE EXTERNLOGIN *login-name*
TO *remote-server*
REMOTE LOGIN *remote-user*
[**IDENTIFIED BY** *remote-password*]

パラメータ

login-name　ローカル・ユーザ・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・ユーザです。

TO 句　リモート・サーバの名前を指定します。

REMOTE LOGIN 句　REMOTE LOGIN 句は、ローカル・ユーザ *login-name* に対して、*remote-server* 上にユーザ・アカウントを指定します。

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。

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・サーバに認識させるには、ISYSSERVER テーブルのエントリを指定します。「CREATE SERVER 文」 444 ページを参照してください。

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 権限を持つユーザのみです。

Windows CE 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」 517 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

サーバ sybase1 に接続するときに、ローカル・ユーザ DBA をパスワード Plankton が設定されたユーザ sa にマップします。

```
CREATE EXTERNLOGIN DBA
TO sybase1
REMOTE LOGIN sa
IDENTIFIED BY Plankton;
```

CREATE FUNCTION 文

この文は、データベース内に新しい関数を作成するために使用します。

構文 1

CREATE [**TEMPORARY**] **FUNCTION** [*owner*.]*function-name* **(** [*parameter*, ⋯] **)**
RETURNS *data-type routine-characteristics*
{ *compound-statement*
  | **AS** *tsql-compound-statement*
  | *external-name* }

構文 2

CREATE FUNCTION [*owner*.]*function-name* **(** [*parameter*, ⋯] **)**
RETURNS *data-type*
URL *url-string*
[**HEADER** *header-string*]
[**SOAPHEADER** *soap-header-string*]
[**TYPE** { **'HTTP**[**:**{ **GET** | **POST** }] **'** | **'SOAP**[**:**{ **RPC** | **DOC** }]**'** }]
[**NAMESPACE** *namespace-string*]
[**CERTIFICATE** *certificate-string*]
[**CLIENTPORT** *clientport-string*]
[**PROXY** *proxy-string*]

url-string :
'{ **HTTP**
  | **HTTPS**
  |**HTTPS_FIPS**}**://**[*user***:***password***@**]*hostname*[**:***port*] }[*/path*]**'**

parameter :
IN *parameter-name data-type* [**DEFAULT** *expression*]

routine-characteristics
ON EXCEPTION RESUME | [**NOT**] **DETERMINISTIC**

tsql-compound-statement:
sql-statement
sql-statement
 ...

external-name:
  EXTERNAL NAME *library-call*
| **EXTERNAL NAME** *java-call* **LANGUAGE JAVA**

library-call :
[*operating-system*:]*function-name***@***library*; ⋯

operating-system :
NetWare | **Unix**

java-call :
[*package-name*.]*class-name.method-name method-signature*

method-signature :
([*field-descriptor*, ⋯]**)** *return-descriptor*

field-descriptor | *return-descriptor* :
Z | **B** | **S** | **I** | **J** | **F** | **D** | **C** | **V** | **[***descriptor* | **L***class-name***;**

パラメータ

CREATE FUNCTION 句　パラメータ名は、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。

関数を実行するときに、すべてのパラメータを指定する必要はありません。CREATE FUNCTION 文の中にデフォルト値がある場合、不明のパラメータ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。

TEMPORARY (CREATE TEMPORARY FUNCTION) を指定すると、作成した接続でのみ表示できる関数になります。また、接続を削除すると、自動的に関数も削除されます。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) のパーミッションで実行されます。また作成者のみが所有できます。そのため、テンポラリ関数を作成するときは*所有者*を指定しないでください。

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。

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。「BEGIN 文」 356 ページを参照してください。

tsql-compound-statement　Transact-SQL 文のバッチ。「Transact-SQL のバッチの概要」『SQL Anywhere サーバ - SQL の使用法』と 「CREATE PROCEDURE 文 [T-SQL]」 434 ページを参照してください。

EXTERNAL NAME 句　EXTERNAL NAME 句を使用する関数は、外部ライブラリ関数呼び出しのラッパです。EXTERNAL NAME を使用する関数では、RETURNS 句の後に他の句を置くことはできません。*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*、NetWare では *.nlm* 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。NetWare では、NLM 名がない場合、関数が呼び出されるときに、記号を含む NLM がロードされます。

外部ライブラリ呼び出しの詳細については、「プロシージャからの外部ライブラリの呼び出し」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

EXTERNAL NAME LANGUAGE JAVA 句　LANGUAGE JAVA 句が付いた EXTERNAL NAME を使用する関数は、Java メソッドのラッパです。

Java プロシージャの呼び出しについては、「CREATE PROCEDURE 文」 423 ページを参照してください。

ON EXCEPTION RESUME 句　Transact-SQL のようなエラー処理を使用します。「CREATE PROCEDURE 文」 423 ページを参照してください。

NOT DETERMINISTIC 句　NOT DETERMINISTIC として指定された関数は、クエリで呼び出されるたびに再評価されます。このような方法で指定されていない関数の結果は、パフォーマンスを向上させるためにキャッシュされ、クエリの評価中に関数が同じパラメータで呼び出されるたびに再利用されます。

基本となるデータを修正する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。このソフトウェアの将来のバージョンでは、この宣言を使用して、同じ入力に対して異なる値を返す可能性のある関数にとっては安全でない最適化でも、実行することが可能になります。

URL 句　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。Web サービスの URL 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」ヘッダに渡します。

HTTPS_FIPS を指定すると、強制的に FIPS ライブラリが使用されます。HTTPS_FIPS 指定したときに、FIPS ライブラリがない場合、代わりに FIPS 以外のライブラリが使用されます。

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 要求ヘッダのエントリを追加または変更します。HTTP ヘッダ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。この句の使用法の詳細については、「CREATE PROCEDURE 文」 423 ページの HEADER 句の説明を参照してください。

HTTP ヘッダの使用の詳細については、「HTTP ヘッダの使用」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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・エントリを指定します。SOAP ヘッダは、静的定数として宣言したり、代入パラメータ・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を定義できますが、INOUT または OUT 代入パラメータは定義できません。この句の使用法の詳細については、「CREATE PROCEDURE 文」 423 ページの SOAPHEADER 句の説明を参照してください。

SOAP ヘッダの使用の詳細については、「SOAP ヘッダの使用」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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、デフォルトのタイプである SOAP:RPC が使用されます。HTTP は HTTP:POST を暗黙的に指定します。SOAP 要求は常に XML 文書として送信されるため、SOAP 要求の送信には常に HTTP:POST が使用されます。

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・サーバから使用できる SOAP サービスの WSDL 記述から取得できます。デフォルト値は、プロシージャの 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。証明書はファイルに置かれ、file キーを使用して提供されるファイルの名前、または証明書全体を文字列に配置できますが、両方は配置できません。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門
name		証明書で指定された共通名

証明書は、HTTPS サーバに対する要求、または安全でないサーバから安全なサーバ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。

CLIENTPORT 句　HTTP クライアント・プロシージャが TCP/IP を使用して通信するポート番号を識別します。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。

「ClientPort プロトコル・オプション [CPORT]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

PROXY 句　プロキシ・サーバの 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。プロシージャがプロキシ・サーバに接続し、そのプロキシ・サーバを介して Web サービスに要求を送信することを示します。

備考

CREATE FUNCTION 文は、データベース内でユーザ定義関数を作成します。所有者名を指定すれば、別のユーザに対する関数を作成できます。パーミッションに従って、ユーザ定義関数を他の非集合関数とまったく同じ方法で使用できます。

SQL Anywhere は、NOT DETERMINISTIC を宣言する場合以外は、すべてのユーザ定義関数を決定的として扱います。決定的関数は、同じパラメータ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・サーバは、同じパラメータ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、クエリの意味に不要な弊害は生じないものとみなします。

関数が結果セットを返す場合、出力パラメータを設定したり戻り値を返したりすることはできません。

Web サービス・クライアント関数の場合、SOAP 関数と HTTP 関数の戻り型は、VARCHAR などの文字データ型の 1 つです。返される値は、HTTP 応答の本文です。HTTP ヘッダ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を使用します。

パラメータ値は、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は、常に要求本文にバンドルされます。

パーミッション

テンポラリ関数を作成しない場合、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。

参照

- 「ALTER FUNCTION 文」 313 ページ
- 「CREATE PROCEDURE 文」 423 ページ
- 「DROP 文」 512 ページ
- 「BEGIN 文」 356 ページ
- 「CREATE PROCEDURE 文」 423 ページ
- 「RETURN 文」 653 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** 永続的ストアド・モジュール機能。

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、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' );
```

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
  DECLARE @Result INT
  SELECT @Result = @Input * 2
  RETURN @Result
```

文 SELECT DoubleIt(5) は、10 の値を返します。

次の文は、Java で作成された外部関数を作成します。

```
CREATE FUNCTION encrypt( IN name char(254) )
RETURNS VARCHAR
EXTERNAL NAME
  'Scramble.encrypt (Ljava/lang/String;)Ljava/lang/String;'
LANGUAGE JAVA;
```

CREATE INDEX 文

この文を使用して、指定されたテーブルまたは実体化ビュー (Materialized View) 上にインデックスを作成します。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。

構文 1 - テーブルにインデックスを作成する

CREATE [**VIRTUAL**] [**UNIQUE**] [**CLUSTERED**] **INDEX** *index-name*
 ON [*owner.*]*table-name*
  (*column-name* [**ASC** | **DESC**], ⋯
   | *function-name* **(** *argument*, ⋯] **) AS** *column-name* **)**
 [{ **IN** | **ON** } *dbspace-name*]
 [**FOR OLAP WORKLOAD**]

構文 2 - 実体化ビュー (Materialized View) にインデックスを作成する

CREATE [**VIRTUAL**] [**UNIQUE**] [**CLUSTERED**] **INDEX** *index-name*
  ON [*owner.*]*materialized-view-name*
  (*column-name* [**ASC** | **DESC**], ⋯**)**
 [{ **IN** | **ON** } *dbspace-name*]
 [**FOR OLAP WORKLOAD**]

パラメータ

VIRTUAL キーワード　VIRTUAL キーワードは主に、Index Consultant によって使用されます。クエリ・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。

「インデックス・コンサルタントの使用」『SQL Anywhere サーバ - SQL の使用法』と「インデックス・コンサルタント」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

CLUSTERED キーワード　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。

「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

UNIQUE キーワード　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実体化ビュー (Materialized View) 内に複数存在しないようにします。各インデックス・キーはユニークであるか、少なくとも 1 つのカラムで NULL を持つ必要があります。

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、または一意性制約があるカラムには、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。

ASC | DESC キーワード　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 句　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・テーブルまたは実体化ビュー (Materialized View) では使用できません。

このフォームの CREATE INDEX 文は、次の操作を実行する際に便利な方法です。

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・タイプは、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。

計算カラムの詳細については、「計算カラムの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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実体化ビュー (Materialized View) 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。

制限事項の詳細については、「SQL Anywhere のサイズと数の制限」『SQL Anywhere サーバ - データベース管理』を参照してください。

FOR OLAP WORKLOAD オプション　FOR OLAP WORKLOAD を指定すると、データベース・サーバ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改善することができます。特に、optimization_workload を OLAP に設定すると有効です。「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

OLAP の詳細については、「OLAP のサポート」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

備考

構文 1 はテーブルで使用し、構文 2 は実体化ビュー (Materialized View) で使用します。

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、作成したインデックスは論理インデックスと扱われます。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。「論理インデックスを使用したインデックスの共有」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

CREATEINDEX 文は、指定されたテーブルまたは実体化ビュー (Materialized View) 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実体化ビュー (Materialized View) 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**　実体化ビュー (Materialized View) 上にはインデックスを作成できますが、実体化ビュー (Materialized View) 以外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実体化ビュー (Materialized View) 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実体化ビュー (Materialized View) 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。

パーミッション

テーブルまたは実体化ビュー (Materialized View) 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。

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。「スナップショット・アイソレーション」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

関連する動作

オートコミット。組み込み関数でインデックスを作成すると、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)。

参照

- 「DROP 文」 512 ページ
- 「インデックス」 『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE STATISTICS 文」 452 ページ
- 「論理インデックスを使用したインデックスの共有」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

Employee 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

プロシージャ内でこの文を使用して、プロシージャが完了した後も、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

column-definition、*column-constraint*、*table-constraint*、*pctfree* の定義については、「CREATE TABLE 文」 460 ページを参照してください。

ON COMMIT　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　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 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」 460 ページ
- 「DECLARE LOCAL TEMPORARY TABLE 文」 495 ページ
- 「複合文の使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL／基本機能。

例

次の例は、ストアド・プロシージャ内のテンポラリ・テーブルの作成方法を示します。

```
BEGIN
  CREATE LOCAL TEMPORARY TABLE TempTab ( number INT );
  ...
END
```

CREATE MATERIALIZED VIEW 文

この文を使用して、実体化ビュー (Materialized View) を作成します。

構文

CREATE MATERIALIZED VIEW
[*owner*.]*materialized-view-name* [**(** *column-name*, … **)**]
[**IN** *dbspace-name*]
AS *select-statement*

パラメータ

column-name リスト　実体化ビュー (Materialized View) に作成するカラムを指定します。*column-name* リストが指定されていない場合、AS 句の *select-statement* で指定されたカラムにカラム名が設定されます。

IN 句　実体化ビュー (Materialized View) を作成する DB 領域を指定します。指定されていない場合は、現在の DB 領域が使用されます。

AS 句　*select-statement* を使用して実体化ビュー (Materialized View) の構造を定義します。実体化ビュー (Materialized View) 定義は、ベース・テーブルのみを参照できます。ビュー、他の実体化ビュー (Materialized View)、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、alias-name を指定します (「SELECT 文」 669 ページを参照してください)。SELECT * 構造体を使用してカラム名を指定することはできません。たとえば、CREATE MATERIALIZED VIEW matview AS SELECT * FROM *table-name* … という内容は指定できません。また、*select-statement* のすべてのオブジェクトは、データベース内でユニークな名前を持っている必要があります。

「実体化ビュー (Materialized View) を管理するときの制限」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

備考

実体化ビュー (Materialized View) は、作成時にデータで自動的に初期化されることはありません。実体化ビュー (Materialized View) を初期化するには、REFRESH MATERIALIZED VIEW 文を使用して個々の実体化ビュー (Materialized View) を初期化するか、sa_refresh_materialized_views システム・プロシージャを使用してデータベース内で初期化されていないすべての実体化ビュー (Materialized View) を初期化します。「REFRESH MATERIALIZED VIEW 文」 640 ページと 「sa_refresh_materialized_views システム・プロシージャ」 941 ページを参照してください。

実体化ビュー (Materialized View) を暗号化し、PCTFREE 設定を変更し、オプティマイザによって使用を有効または無効にすることができます。ただし、実体化ビュー (Materialized View) を作成してから、ALTER MATERIALIZED VIEW を使用してこれらのオプションを設定します。これらのオプションを作成したときのデフォルト値は、データベースに使用するページ・サイズに応じて (ページ・サイズが 4 KB の場合は 200 バイト、2 KB の場合は 100 バイト)、NOT ENCRYPTED、ENABLE USE IN OPTIMIZATION、PCTFREE (デフォルト) です。

sa_recompile_views システム・プロシージャは、実体化ビュー (Materialized View) を再コンパイルしません。

実体化ビュー (Materialized View) を作成するには、一部のオプションに特定の値を指定する必要があります。「実体化ビュー (Materialized View) を管理するときの制限」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

パーミッション

実体化ビュー (Materialized View) 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。別のユーザの実体化ビュー (Materialized View) 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、実体化ビュー (Materialized View) 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

- 「実体化ビュー (Materialized View) の編集」『SQL Anywhere サーバ - SQL の使用法』
- 「ALTER MATERIALIZED VIEW 文」 316 ページ
- 「DROP 文」 512 ページ
- 「REFRESH MATERIALIZED VIEW 文」 640 ページ
- 「CREATE VIEW 文」 482 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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実体化ビュー (Materialized View) を作成します。この後、ビューを初期化して使用するには、REFRESH MATERIALIZED VIEW 文を実行する必要があります。

```
CREATE MATERIALIZED VIEW EmployeeConfidential AS
SELECT EmployeeID, Employees.DepartmentID,
   SocialSecurityNumber, Salary, ManagerID,
   Departments.DepartmentName, Departments.DepartmentHeadID
FROM Employees, Departments
WHERE Employees.DepartmentID=Departments.DepartmentID
ORDER BY Employees.DepartmentID;
```

CREATE MESSAGE 文 [T-SQL]

この文は、PRINT 文と RAISERROR 文によって使用される、ユーザ定義メッセージをシステム・テーブル ISYSUSERMESSAGE に追加する場合に使用します。

構文

CREATE MESSAGE *message-number* **AS** *message-text*

message-number : *integer*

message-text : *string*

パラメータ

message_number　追加するメッセージのメッセージ番号。ユーザ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。

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、プレースホルダには番号が付けられます。引数のプレースホルダのフォームは "%nn!" です。パーセント記号 (%)、nn (1 ー 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。

メッセージを削除する方法については、「DROP 文」 512 ページを参照してください。

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]」 632 ページ
- 「RAISERROR 文 [T-SQL]」 635 ページ
- 「ISYSUSERMESSAGE システム・テーブル」 760 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

CREATE PROCEDURE 文

この文は、データベース内にプロシージャを作成するために使用します。

構文 1 - 定義されたプロシージャの作成

CREATE [**TEMPORARY**] **PROCEDURE** [*owner*.]*procedure-name* **(** [*parameter*, ⋯] **)**
{ [**RESULT (** *result-column*, ⋯ **)** | **NO RESULT SET**]
[**ON EXCEPTION RESUME**]
compound-statement
| **AT** *location-string*
| **EXTERNAL NAME** *library-call*
| [**DYNAMIC RESULT SETS** *integer-expression*]
[**EXTERNAL NAME** *java-call* **LANGUAGE JAVA**]
}

parameter :
parameter-mode parameter-name data-type [**DEFAULT** *expression*]
| **SQLCODE**
| **SQLSTATE**

parameter-mode : **IN** | **OUT** | **INOUT**

result-column : *column-name data-type*

library-call :
[*operating-system*:]*function-name*@*library*; ⋯

operating-system : **NetWare** | **Unix**

java-call :
[*package-name*.]*class-name.method-name method-signature*

method-signature :
([*field-descriptor*, ⋯] **)** *return-descriptor*

field-descriptor | *return-descriptor* :
Z | **B** | **S** | **I** | **J** | **F** | **D** | **C** | **V** | **[***descriptor* | **L***class-name***;**

構文 2 - Web サービスの作成

CREATE PROCEDURE [*owner*.]*procedure-name* **(** [*parameter*, ⋯] **)**
URL *url-string*
[**HEADER** *header-string*]
[**SOAPHEADER** *soap-header-string*]
[**TYPE** { **'HTTP**[:{ **GET** | **POST** }]**'** | **'SOAP**[:{ **RPC** | **DOC** }]**'** }]
[**NAMESPACE** *namespace-string*]
[**CERTIFICATE** *certificate-string*]
[**CLIENTPORT** *clientport-string*]
[**PROXY** *proxy-string*]
[**SET** *protocol-option-string*

parameter :
parameter-mode parameter-name data-type [**DEFAULT** *expression*]
| **SQLCODE**
| **SQLSTATE**

parameter-mode : **IN** | **OUT** | **INOUT**

url-string :
{ **HTTP** | **HTTPS** | **HTTPS_FIPS** }**://**[*user*:*password*@]*hostname*[**:***port*][**/***path*]

header-string :
The string to use for the HTTP header.

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*
)

soap-operation-name :
The name of the SOAP operation to call.

パラメータ

CREATE PROCEDURE 句　永続的またはテンポラリ (TEMPORARY) のストアド・プロシージャを作成できます。PROC は PROCEDURE の同義語として使用できます。

パラメータ名は、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 (「SQL データ型」 47 ページを参照してください)。パラメータには、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**　このパラメータは、プロシージャに値を与える式です。
- **OUT**　このパラメータは、プロシージャから値を受け取ることがある変数です。
- **INOUT**　パラメータはプロシージャに値を与え、プロシージャ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を実行する場合、必ずしもすべてのパラメータ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、プロシージャが終了するときに、SQLSTATE または SQLCODE 値を出力する特別なパラメータです (OUT パラメータです)。SQLSTATE と SQLCODE パラメータを指定しても、指定しなくても、SQLSTATE と SQLCODE 特別値をプロシージャ・コールのすぐ後でチェックして、プロシージャのリターン・ステータスをテストします。

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。

TEMPORARY (CREATE TEMPORARY PROCEDURE) を指定すると、作成した接続でのみ表示できるストアド・プロシージャになります。また、接続を削除すると、自動的に関数も削除されます。テンポラリ・ストアド・プロシージャ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・ストアド・プロシージャは、作成者 (現在のユーザ) のパーミッションで実行されます。また作成者のみが所有できます。そのため、テンポラリ・ストアド・プロシージャを作成するときは*所有者*を指定しないでください。

読み込み専用データベースに接続し、外部プロシージャ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 CustRank char(128) )
BEGIN
   IF EXISTS ( SELECT * FROM SYS.SYSTAB WHERE table_name = CustRank ) THEN
  EXECUTE IMMEDIATE 'DROP TABLE "' || CustRank || '"';
  MESSAGE 'Table "' || CustRank || '" dropped' to client;
   END IF;
END;
```

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。許容されるデータ型のリストについては、「SQL データ型」 47 ページを参照してください。

プロシージャから返される結果セットの詳細については、「プロシージャから返される結果」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

プロシージャは、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・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・プロシージャは、これらのインタフェー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、ドライバまたはプロバイダ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・プロシージャは Open Client アプリケーションで使用できます。

プロシージャが結果セットを 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。

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で使用可能にします。

ON EXCEPTION RESUME を使用すると、プロシージャは 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。

「on_tsql_error オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

AT location-string 句　*location-string* に指定されたリモート・プロシージャのプロキシ・ストアド・プロシージャ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・プロシージャは出力変数に最大 254 文字まで返すことができます。

リモート・サーバについては、「CREATE SERVER 文」 444 ページを参照してください。リモート・プロシージャの使用については、「リモート・プロシージャ・コール (RPC) の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

EXTERNAL NAME 句　EXTERNAL NAME 句を使用するプロシージャは、外部ライブラリの呼び出しのラッパです。EXTERNAL NAME を使用するストアド・プロシージャでは、パラメータ・リストの後に他の句を置くことはできません。*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*、NetWare では *.nlm* 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。NetWare では、NLM 名がない場合、関数が呼び出されるときに、記号を含む NLM がロードされます。

外部ライブラリ呼び出しの詳細については、「プロシージャからの外部ライブラリの呼び出し」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

DYNAMIC RESULT SETS 句　この句は直接 EXTERNAL NAME LANGUAGE JAVA 句に関連付けられます。また、Java メソッドをラップするプロシージャで使用します。DYNAMIC RESULT SETS 句を指定しない場合、メソッドは結果セットを返さないとみなされます。

EXTERNAL NAME java-call LANGUAGE JAVA 句　LANGUAGE JAVA 句が付いた EXTERNAL NAME を使用するプロシージャは、Java メソッドのラッパです。Java メソッド・シグニチャは、パラメータの型と戻り値の型を簡潔に文字で表現したものです。パラメータの数がメソッド・シグニチャに指定された数字よりも小さい場合は、この差が DYNAMIC RESULT SETS に指定された数と等しくなるようにします。また、プロシージャ・パラメータ・リストよりも多いメソッド・シグニチャ内の各パラメータには、メソッド・シグニチャ [Ljava/SQL/ResultSet; が必要です。

field-descriptor と *return-descriptor* の意味は次のとおりです。

フィールド・タイプ	Java データ型
B	byte
C	char
D	double
F	float
I	int
J	long

フィールド・タイプ	Java データ型
L *class-name*;	クラス *class-name* のインスタンス。クラス名は、完全に修飾された名前で、ドットを / に置き換えたものとします。例：java/lang/String
S	short
V	void
Z	ブール式
[	配列の各次元ごとに 1 つ使用

次に例を示します。

```
double some_method(
  boolean a,
  int b,
  java.math.BigDecimal c,
  byte [][] d,
  java.sql.ResultSet[] rs ) {
}
```

この例では、次のシグニチャを得られます。

```
'(ZILjava/math/BigDecimal;[[B[Ljava/SQL/ResultSet;)D'
```

「Java メソッドから返される結果セット」『SQL Anywhere サーバ - プログラミング』を参照してください。

URL 句　HTTP または SOAP Web サービス・クライアント・プロシージャを定義する場合にのみ使用します。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 ヘッダに渡します。

HTTPS_FIPS を指定すると、強制的に FIPS ライブラリが使用されます。HTTPS_FIPS 指定したときに、FIPS ライブラリがない場合、代わりに FIPS 以外のライブラリが使用されます。

この方法で指定すると、ユーザ名とパスワードは URL の一部として暗号化されずに渡されます。

HEADER 句
HTTP Web サービス・クライアント・プロシージャを作成する場合は、この句を使用して、HTTP 要求ヘッダのエントリを変更または追加、既存のヘッダの抑制を行います。ヘッダの仕様は、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。HTTP ヘッダ仕様の重要な点を次に示します。

- ヘッダ／値ペアは、¥n または ¥x0d¥n で区切ります。これは、それぞれ改行 (<LF>) または復帰と改行 (<CR><LF>) です。

- ヘッダと値は、コロン (:) を使用して区切ります。このため、ヘッダにコロンは使用できません。
- 直後に :¥n または行の最後があるヘッダは、値のないヘッダを表します。コロンや値のないヘッダも同様です。たとえば、HEADER 'Date' は、含まれない Date ヘッダであることを指定します。ヘッダや値を抑制すると、予期しない結果になる場合があります。「HTTP ヘッダの修正」『SQL Anywhere サーバ - プログラミング』を参照してください。
- 長いヘッダ値の折り返しがサポートされています。折り返すには、¥n の直後に 1 つ以上のスペースが必要です。たとえば、次の HEADER 仕様とこれによる HTTP 出力は、セマンティック上は同じです。

```
... HEADER 'heading1: This long value¥n is a really long value for heading1¥n
heading2:shortvalue'

heading1:This long value is a really long value for heading1<CR><LF>
heading2:shortvalue<CR><LF>
```

- 連続する複数のスペースは、折り返しの場合も含めて、単一の空白文字として出力されます。
- この句では、パラメータの代入がサポートされています。

次の例は、静的なユーザ定義ヘッダの追加方法を示しています。

```
CREATE PROCEDURE http_client() URL 'http://localhost/getdata'
  TYPE 'http:get' HEADER 'UserHeader1:value1¥nUserHeader2:value2';
```

次の例は、代入パラメータを使用する新しいユーザ定義ヘッダの追加方法を示しています。

```
CREATE PROCEDURE http_client( headers LONG VARCHAR ) URL 'http://localhost/getdata'
TYPE 'http:get' HEADER '!headers';
CALL http_client( 'NewHeader1:value1¥nNewHeader2:value2' );
```

HTTP ヘッダの使用の詳細については、「HTTP ヘッダの使用」『SQL Anywhere サーバ - プログラミング』を参照してください。

SOAPHEADER 句　SOAP Web サービスをプロシージャ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・エントリを指定します。SOAP ヘッダは、静的定数として宣言したり、代入パラメータ・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 パラメータを宣言)。Web サービス・プロシージャでは、1 つ以上の IN モード代入パラメータと、1 つの INOUT または OUT 代入パラメータを定義できます。

次の例は、クライアントが、いくつかのヘッダ・エントリをパラメータを使用して送信し、応答 SOAP ヘッダ・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 VARCHAR hd1, IN VARCHAR hd2, IN VARCHAR hd3 )
  URL 'localhost/some_endpoint'
  SOAPHEADER '!hd1!hd2!hd3';
```

SOAP ヘッダの使用の詳細については、「SOAP ヘッダの使用」『SQL Anywhere サーバ - プログラミング』を参照してください。

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、デフォルトのタイプである SOAP:RPC が使用されます。HTTP は HTTP:POST を暗黙的に指定します。SOAP 要求は常に XML 文書として送信されるため、SOAP 要求の送信には常に HTTP:POST が使用されます。

NAMESPACE 句　SOAP クライアント・プロシージャ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・サーバから使用できる SOAP サービスの WSDL 記述から取得できます。デフォルト値は、プロシージャの 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。証明書はファイルに置かれ、file キーを使用して提供されるファイルの名前、または証明書全体を文字列に配置できますが、両方は配置できません。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門
name		証明書で指定された共通名

証明書は、HTTPS サーバに対する要求、または安全でないサーバから安全なサーバ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。

CLIENTPORT 句　HTTP クライアント・プロシージャが TCP/IP を使用して通信するポート番号を識別します。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。

「ClientPort プロトコル・オプション [CPORT]」『SQL Anywhere サーバ - データベース管理』を参照してください。

PROXY 句　プロキシ・サーバの 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。プロシージャがプロキシ・サーバに接続し、そのプロキシ・サーバを介して 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、パラメータの代入がサポートされています。

- **CH または CHUNK**　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 2048 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。チャンクが、AUTO モードのときにステータス 505 ('サポートされていない HTTP バージョン') で失敗するか、ステータス 501 ('実装されていません') で失敗した場合は、クライアントはチャンクされたエンコードなしで要求をリトライします。

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。

- **VER または VERSION**　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

- **OP または OPERATION**　このオプションでは、SOAP 処理の名前を指定します (作成しているプロシージャの名前と異なる場合)。OPERATION の値は、リモート・プロシージャ・コールの名前に似ています。たとえば、login という SOAP 処理を呼び出す accounts_login というプロシージャを作成するとき、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 処理の名前は、作成するプロシージャの名前と一致しなければなりません。

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 )'
  ...
```

例を含む Web サービスを作成する詳細については、「SQL Anywhere Web サービス」 『SQL Anywhere サーバ - プログラミング』を参照してください。

備考

CREATE PROCEDURE 文はデータベースにプロシージャを作成します。DBA 権限があるユーザは、所有者を指定することによって他のユーザのプロシージャを作成できます。プロシージャは CALL 文で呼び出します。

ストアド・プロシージャが結果セットを返す場合、出力パラメータを設定したり戻り値を返したりすることはできません。

Web サービス・クライアント・プロシージャの場合、パラメータ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は、常に要求本文にバンドルされます。

パーミッション

テンポラリ・プロシージャを作成しない場合、RESOURCE 権限が必要です。

別のユーザのプロシージャを作成するには、外部プロシージャの DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「BEGIN 文」 356 ページ
- 「CALL 文」 362 ページ
- 「CREATE FUNCTION 文」 408 ページ
- 「CREATE PROCEDURE 文 [T-SQL]」 434 ページ
- 「ALTER PROCEDURE 文」 319 ページ
- 「DROP 文」 512 ページ
- 「EXECUTE IMMEDIATE 文 [SP]」 536 ページ
- 「GRANT 文」 565 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** 永続的ストアド・モジュール機能。Java 結果セットの構文拡張は、オプションの J621 機能に指定されています。

例

次のプロシージャは、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'
  WHEN 'Shorts' THEN
    SET type = 'Shorts'
  ELSE
    SET type = 'UNKNOWN'
  END CASE;
END;
```

次のプロシージャ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 PROCEDURE 文 [T-SQL]

この文は、Adaptive Server Enterprise 互換の方法で、データベース内にプロシージャを作成するために使用します。

構文 1

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 PROCEDURE [*owner.*]*procedure_name*
[**NO RESULT SET**]
[[**(**] *@parameter_name data-type* [= *default*] [**OUTPUT**], ⋯ [**)**]]
[**WITH RECOMPILE**] **AS** *statement-list*

パラメータ

NO RESULT SET 句　このプロシージャ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。

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際に役立ちます。

- **@ をプレフィクスとする変数名**　"@" 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**　Watcom-SQL プロシージャ・パラメータは、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・パラメータは、デフォルトでは INPUT パラメータであり、OUTPUT としても指定できます。これらの SQL Anywhere で INOUT または OUT として宣言されるパラメータは、Transact-SQL では OUTPUT として宣言してください。
- **パラメータのデフォルト値**　Watcom-SQL のプロシージャ・パラメータには、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。
- **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 プロシージャは、結果セットが Transact-SQL ストアド・プロシージャ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;
```

- **プロシージャ本体**　Transact-SQL プロシージャの本体は、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 プロシージャの本体は、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 PROCEDURE 文」 423 ページ

標準と互換性

- **SQL/2003**　Transact-SQL 拡張。
- **Sybase**　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。

  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を再コンパイルし、コンパイルされたプロシージャをデータベースが停止するまで保管します。

  プロシージャのグループはサポートされません。

CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]

この文は、パブリケーションを作成するために使用します。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 [*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 [*owner.*]*publication-name*
WITH SCRIPTED UPLOAD
(*article-definition*, ⋯ **)**

article-definition :
  TABLE *table-name* [**(** *column-name*, ⋯ **)**]
[**USING (** [**PROCEDURE**] [*owner.*][*procedure-name*]
  FOR UPLOAD { **INSERT** | **DELETE** | **UPDATE** }, ... **)**]

構文 3 (Mobile Link のダウンロードのみパブリケーション)

CREATE PUBLICATION [*owner.*]*publication-name*
FOR DOWNLOAD ONLY
(*article-definition*, ⋯ **)**

article-definition : **TABLE** *table-name* [**(** *column-name*, ⋯ **)**]

構文 4 (SQL Remote)

CREATE PUBLICATION [*owner.*]*publication-name*
(*article-definition*, ⋯ **)**

article-definition :
  TABLE *table-name* [**(** *column-name*, ⋯ **)**]
[**WHERE** *search-condition*]
[**SUBSCRIBE BY** *expression*]

パラメータ

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。複数のアーティクルを含めるには、アーティクル間をカンマで区切ります。各アーティクルは、テーブルまたはテーブルの一部です。アーティクルは、テーブルの縦の分割 (テーブル・カラムのサブセット)、横の分割 (WHERE 句に基づいたテーブル・ローのサブセット)、または縦横の分割の場合があります。

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。

「スクリプト化されたアップロードのパブリケーションの作成」『Mobile Link - クライアント管理』を参照してください。

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 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します。

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 (挿入、削除、更新用のプロシージャを 1 つずつ) を使用できます。

構文 3 は、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

DBA 権限が必要です。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

- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」 321 ページ
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」 518 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 455 ページ
- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 332 ページ
- SQL Anywhere Mobile Link クライアント: 「データのパブリッシュ」 『Mobile Link - クライアント管理』
- Ultra Light Mobile Link クライアント :「Ultra Light CREATE PUBLICATION 文」 『Ultra Light - データベース管理とリファレンス』
- SQL Remote :「データのパブリッシュ」 『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」 759 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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、Customer 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 customer 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 )
  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]

この文は、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・アドレスの識別に使用します。

構文

CREATE REMOTE MESSAGE TYPE *message-system*
ADDRESS *address*

message-system: **FILE** | **FTP** | **MAPI** | **SMTP** | **VIM**

address: *string*

パラメータ

message-system　サポートしているメッセージ・システムの名前。

address　指定したメッセージ・システムのアドレス。

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ディレクトリを指定します。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。

初期化ユーティリティは、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。

> **注意**
> VIM および MAPI のサポートは廃止されました。

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

- 「GRANT PUBLISH 文 [SQL Remote]」 572 ページ

- 「GRANT REMOTE 文 [SQL Remote]」 573 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」 570 ページ
- 「DROP REMOTE MESSAGE TYPE 文 [SQL Remote]」 519 ページ
- 「メッセージ・タイプの使用」 『SQL Remote』

標準と互換性

- **SQL/2003** ベンダ拡張。

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* サブディレクトリに送り返すように設定できます。

また、dbremote に対して、*company* サブディレクトリで入力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

この文は、データベース・ユーザに対して、テーブル、ビュー、パーミッションのコレクションを作成するために使用します。

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 ID を指定します。別のユーザ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、文デリミタで区切りません。文デリミタは 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。

現在は 1 ユーザに対して複数のスキーマを作成することができますが、このような処理はおすすめできませんし、今後のバージョンではサポートされないかもしれません。

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」 460 ページ
- 「CREATE VIEW 文」 482 ページ
- 「GRANT 文」 565 ページ

標準と互換性

- **SQL/2003**　コア機能。
- **Sybase**　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内での使用も許可されていません。

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 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 文の文デリミタは、最初の CREATE TABLE 文の後に配置されます。文デリミタは CREATE SCHEMA 文の終わりをマークするので、例は、データベース・サーバによって 2 つの文バッチとして解釈されます。したがって、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 SERVER 文

この文を使用して、リモート・サーバを作成します。

構文 1

CREATE SERVER *server-name*
CLASS *server-class*
USING *connection-info*
[**READ ONLY**]

server-class :
SAJDBC | **ASEJDBC** | **SAODBC** | **ASEODBC**
| **DB2ODBC** | **MSSODBC** | **ORAODBC** | **ODBC**

connection-info :
{ *computer-name*:*port-number* [*/dbname*] | *data-source-name* | *sqlanywhere-connection-string* }

構文 2

CREATE SERVER *server-name*
CLASS 'DIRECTORY'
USING *using-clause*

using-clause :
ROOT=*path*
[**;SUBDIRS**=*n*]
[**;READONLY**={ **YES** | **NO** }]

パラメータ

CLASS 句 リモート接続に使用するサーバ・クラスを指定します。サーバ・クラスには、詳細なサーバ機能情報が入っています。NetWare を使用している場合、SAJDBC クラスのみサポートされます。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によって決まり、ドライバは *server-class* によって決まります。

JDBC ベースのサーバ・クラスを使用する場合、USING 句のフォームは *hostname*:*portnumber* [*/dbname*] となります。

- **hostname** リモート・サーバが稼働しているコンピュータ。
- **portnumber** リモート・サーバが受信する 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。

ODBC ベースのサーバ・クラスを使用する場合、USING 句は *data-source-name* になります。*data-source-name* は ODBC データ・ソース名です。

SQL Anywhere リモート・サーバ (SAJDBC または SAODBC サーバ・クラス) の場合、*connection-info* パラメータには任意の有効な 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 接続パラメータを含めて接続試行をトラブルシューティングできます。

SQL Anywhere の接続文字列の詳細については、「接続パラメータ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

UNIX プラットフォームでは、ODBC ドライバ・マネージャも参照する必要があります。たとえば、指定された iAnywhere Solutions ODBC ドライバを使用すると、構文は次のようになります。

```
USING 'driver=SQL Anywhere 10;dsn=my_dsn'
```

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** このパスはデータベース・サーバ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** データベース・サーバ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 ー 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・サーバ経由で使用できるディレクトリまたはサブディレクトリのいずれかにプロキシ・テーブルを作成できます。
- **READONLY** ディレクトリ・アクセス・サーバからアクセスするファイルを修正できる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。

備考

リモート・サーバを作成すると、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・サーバを定義します。

サーバ・クラスの詳細とサーバの設定方法については、「リモート・データ・アクセスのサーバ・クラス」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

構文 2 CREATE SERVER 文を使用すると、データベース・サーバが稼働しているコンピュータのローカル・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・サーバを作成できます。ディレクトリ・アクセス・サーバを使用する必要があるデータベース・ユーザごとに、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・サーバは特定のユーザ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・ユーザに付与されているパーミッションに基づきます。

ディレクトリ・アクセス・サーバの詳細については、「ディレクトリ・アクセス・サーバの使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

パーミッション

このコマンドを実行するには、DBA 権限が必要です。

Windows CE 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」 325 ページ
- 「DROP SERVER 文」 520 ページ
- 「リモート・データ・アクセスのサーバ・クラス」 『SQL Anywhere サーバ - SQL の使用法』
- 「ISYSSERVER システム・テーブル」 758 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例は、コンピュータ apple にあり、ポート番号 2638 で受信している Adaptive Server Anywhere リモート・サーバ testasa を作成します。

```
CREATE SERVER testsa
CLASS 'SAJDBC'
USING 'apple:2638';
```

次の例は、JDBC ベースの Adaptive Server のリモート・サーバ ase_prod を作成します。このコンピュータ名は banana でポート番号は 3025 です。

```
CREATE SERVER ase_prod
CLASS 'asejdbc'
USING 'banana:3025';
```

次の例は、Oracle サーバ oracle723 のリモート・サーバを作成します。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

この文を使用すると、データベース・サーバは Web サーバとして動作します。

構文 1 - DISH サービス

CREATE SERVICE *service-name*
TYPE 'DISH'
[**GROUP** { *group-name* | **NULL** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** | **'XML'** | **NULL** }]
[*common-attributes*]

構文 2 - SOAP サービス

CREATE SERVICE *service-name*
TYPE 'SOAP'
[**DATATYPE** { **ON** | **OFF** | **IN** | **OUT** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** | **'XML'** | **NULL** }]
[*common-attributes*]
AS *statement*

構文 3 - その他のサービス

CREATE SERVICE *service-name*
TYPE { **'RAW'** | **'HTML'** | **'XML'** }
[**URL** [**PATH**] { **ON** | **OFF** | **ELEMENTS** }]
[*common-attributes*]
[**AS** { *statement* | **NULL** }]

common-attributes:
[**AUTHORIZATION** { **ON** | **OFF** }]
[**SECURE** { **ON** | **OFF** }]
[**USER** { *user-name* | **NULL** }]

パラメータ

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 /、-、_、.、!、~、*、'、(,) です。ただし、スラッシュ (/) は、先頭の文字には使用できず、また 2 つ以上連続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、DISH サービス名にフォワード・スラッシュ (/) を指定することはできません。

AUTHORIZATION 句　サービスに接続するときに、ユーザ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。認証が OFF の場合、AS 句が必要となり、USER 句によって 1 人のユーザが識別されます。そのユーザ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。

認証が ON の場合、すべてのユーザはユーザ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 句を使用しユーザ名またはグループ名を指定することにより、ユーザに対しサービスの使用許可を制限することもできます。ユーザ名が NULL の場合、すべてのユーザはすべてサービスにアクセスできます。

デフォルト値は ON です。運用システムでは認証を ON にして実行すること、ユーザをグループに追加することによってサービス使用のパーミッションを付与することをおすすめします。

SECURE 句　安全化されていない接続を受け入れるかどうかを示します。ON は、HTTPS 接続のみを受け入れることを示します。HTTP ポートで受信されたサービス要求は、自動的に HTTPS ポートにリダイレクトされます。OFF に設定すると、HTTP 接続と HTTPS 接続の両方を受け入れます。デフォルト値は OFF です。

USER 句　認証を無効にすると、このパラメータは必須となり、すべてのサービス要求を実行する際に使用されるユーザ ID が指定されます。認証を有効 (デフォルト) にすると、このオプション句は、サービスへのアクセスを許可されるユーザまたはグループを識別します。デフォルト値は NULL です。これによって、すべてのユーザにアクセスが付与されます。

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。OFF は、URL 要求のサービス名の後に何も指定する必要がないことを示します。ON は、URI の残りの部分が URL という名前の変数の値として解釈されることを示します。ELEMENTS は、URL パスの残りの部分が、スラッシュで、最大 10 の要素からなるリストに分割されることを示します。値は、url という名前の変数と 1 から 10 までの数値サフィックスに割り当てられます。たとえば、最初の 3 つの変数名は、url1、url2、url3 です。指定される値の数が 10 より少ない場合、残りの変数は NULL に設定されます。サービス名が / で終わる場合、URL を OFF に設定する必要があります。デフォルト値は OFF です。

GROUP 句　DISH サービスにだけ適用されます。DISH サービスがどの SOAP サービスを公開するかを制御する共通プレフィクスを指定します。たとえば、GROUP *xyz* を指定すると、SOAP サービス *xyz*/*aaaa*、*xyz*/*bbbb*、または *xyz*/*cccc* のみ公開され、*abc/aaaa* または *xyzaaaa* は公開されません。グループ名が指定されていない場合、DISH サービスはデータベース内のすべての SOAP サービスを公開します。SOAP サービスは、複数の DISH サービスによって公開される場合があります。サービス名で使用できる文字と同じ文字をグループ名に使用できます。

DATATYPE 句　SOAP サービスにだけ適用されます。すべての SOAP サービス・フォーマットでパラメータ入力または結果セットの出力にデータ入力をサポートするかどうかを制御します。サポートする場合、データ入力では SOAP ツールキットを使用してデータを解析し、適切な型にキャストします。パラメータのデータ型は、DISH サービスが生成する Web サービス定義言語 (WSDL: Web Service Definition Language) 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**　入力パラメータと結果セットの応答のデータ入力をサポートします。
- **OFF**　入力パラメータと結果セットの応答のデータ入力をサポートしません (デフォルト)。
- **IN**　入力パラメータのデータ入力のみをサポートします。
- **OUT**　結果セットの応答のデータ入力のみをサポートします。

SOAP サービスの詳細については、「SOAP サービスの使用」『SQL Anywhere サーバ - プログラミング』を参照してください。

FORMAT 句　DISH および SOAP サービスにだけ適用されます。.NET や Java JAX-RPC などのさまざまなタイプの SOAP クライアントと互換性のある出力フォーマットを生成します。SOAP サービスのフォーマットが指定されていない場合、フォーマットはサービスの DISH サービス宣言から継承されます。DISH サービスでもフォーマットが宣言されていない場合、デフォルトは、.NET クライアントと互換性のある DNET です。フォーマットを宣言しない SOAP サービス

は、複数の DISH サービスを定義することにより、それぞれ異なる FORMAT タイプを持つさまざまな種類の SOAP クライアントで使用できます。

次のフォーマットがサポートされます。

- **DNET** .NET SOAP クライアントに使用する Microsoft DataSet フォーマット。DNET はデフォルトの FORMAT 値であり、バージョン 9.0.2 より前で使用できた唯一のフォーマットです。
- **CONCRETE** プラットフォームに依存しない DataSet フォーマットであり、JAX-RPC などのクライアント、または返されるデータ構造のフォーマットに基づいてインタフェースを自動的に生成するクライアントで使用します。このフォーマット・タイプを指定すると、WSDL 内の SimpleDataset 要素が公開されます。この要素は、結果セットを、それぞれカラム要素の配列を含むローの配列から構成されるローセットの包含階層として記述します。
- **XML** 単純な XML 文字列フォーマット。XML パーサに渡すことのできる文字列として DataSet が返されます。このフォーマットは、SOAP クライアント間で最も移植が簡単です。

TYPE 句 Type 句は、返される結果セットの種類を示します。

サポートされる結果タイプは次のとおりです。

- **RAW** 結果セットがフォーマットされずにクライアントに送られます。要求されたタグをプロシージャ内で明示的に生成することによって、フォーマットされた文書を作成できます。
- **HTML** 文またはプロシージャの結果セットは、テーブルを格納する HTML ドキュメントに自動的にフォーマットされます。
- **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 以外の場合は、自動的に XML としてフォーマットされます。効果は、SELECT 文で FOR XML RAW 句を使用した場合と同様です。
- **SOAP** 結果セットは、SOAP 応答として返されます。データのフォーマットは、FORMAT 句によって決定されます。SOAP サービスへのすべての要求は、単純な HTTP 要求ではなく有効な SOAP 要求である必要があります。SOAP 規格の詳細については、www.w3.org/TR/SOAP を参照してください。
- **DISH** DISH サービス (SOAP ハンドラを決定) は、GROUP 句で識別される SOAP サービスのプロキシとして機能し、これらの各 SOAP サービスに対して WSDL (Web Services Description Language) ファイルを生成します。

  詳細については、「Web サービスの作成」『SQL Anywhere サーバ - プログラミング』を参照してください。

statement statement が NULL の場合、実行する statement を URL で指定する必要があります。そうしないと、指定された SQL 文しかサービスを使って実行されません。この SQL 文は、SOAP サービスでは必須です。デフォルト値は NULL です。

運用システムで実行するすべてのサービスに statement を定義することを強くおすすめします。認証が有効になっている場合のみ、statement に NULL が指定できます。

備考

CREATE SERVICE 文を使用すると、データベース・サーバは Web サーバとして動作します。新しいエントリが ISYSWEBSERVICE システム・テーブルに作成されます。

パーミッション

DBA 権限が必要です。

関連する動作

なし。

参照

- 「ALTER SERVICE 文」 327 ページ
- 「DROP SERVICE 文」 521 ページ
- 「ISYSWEBSERVICE システム・テーブル」 760 ページ
- 「SQL Anywhere Web サービス」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

Web サーバをすばやく設定するには、-xs オプション (たとえば -x http) オプションを使用してデータベース・サーバを起動し、次の文を実行します。

```
CREATE SERVICE tables TYPE 'HTML'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT *
    FROM SYS.SYSTAB;
```

この文を実行したら、Web ブラウザを使用して URL http://localhost/tables を開きます。

次の例は、Hello World プログラムの作成方法を示します。

```
CREATE PROCEDURE hello_world_proc( )
RESULT (html_doc long varchar)
BEGIN
  CALL dbo.sa_set_http_header( 'Content-Type', 'text/html' );
  SELECT '<html>¥n'
      || '<head><title>Hello World</title></head>¥n'
      || '<body>¥n'
      || '<h1>Hello World!</h1>¥n'
      || '</body>¥n'
      || '</html>¥n';
END;

CREATE SERVICE hello_world TYPE 'RAW'
AUTHORIZATION OFF
USER DBA
AS CALL hello_world_proc;
```

この文を実行したら、Web ブラウザを使用して URL *http://localhost/hello_world* を開きます。

CREATE STATISTICS 文

オプティマイザ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 *table-name* [**(** *column-list* **)**]

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、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します。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。

CREATE STATISTICS 文を使用する状況はまれです。

パーミッション

DBA 権限が必要です。

関連する動作

クエリ・プランは変化する可能性があります。

参照

- 「DROP STATISTICS 文」 523 ページ
- 「ALTER DATABASE 文」 303 ページ
- 「LOAD TABLE 文」 603 ページ
- 「クエリの最適化と実行」 『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE INDEX 文」 414 ページ
- 「ISYSCOLSTAT システム・テーブル」 753 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

CREATE SUBSCRIPTION 文 [SQL Remote]

この文は、パブリケーションに対するユーザのサブスクリプションの作成に使用します。

構文

CREATE SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*

publication-name: *identifier*

subscription-value, *subscriber-id*: *string*

パラメータ

publication-name　ユーザのサブスクリプションを作成するパブリケーション名。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のユーザ ID。ユーザには、REMOTE パーミッションが必要です。

備考

SQL Remote インストール環境では、データはレプリケート用にパブリケーション単位で編成されます。SQL Remote メッセージを受信するには、REMOTE パーミッションを持つユーザ ID でサブスクリプションを作成します。

サブスクリプションに指定されている文字列が、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。

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・データベース上の統合データベースにもサブスクリプションを作成してください。この作業は、抽出ユーティリティが自動的に実行します。

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

- 「DROP SUBSCRIPTION 文 [SQL Remote]」 524 ページ
- 「GRANT REMOTE 文 [SQL Remote]」 573 ページ
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」 714 ページ
- 「START SUBSCRIPTION 文 [SQL Remote]」 703 ページ

- 「ISYSSUBSCRIPTION システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、パブリケーション pub_sales にユーザ 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 SUBSCRIPTION 文 [Mobile Link]

SQL Anywhere リモート・データベースでこの文を使用して、Mobile Link ユーザ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
TO *publication-name*
[**FOR** *ml_username*, …]
[**TYPE** *network-protocol*]
[**ADDRESS** *protocol-options*]
[**OPTION** *option*=*value*, …]

ml_username: *identifier*

network-protocol: **http** | **https** | **tls** | **tcpip**

protocol-options: *string*

value: *string* | *integer*

パラメータ

TO 句 パブリケーション名を指定します。

FOR 句 1 つ以上の Mobile Link ユーザ名を指定します。ユーザ名を複数指定した場合、ユーザごとに別個のサブスクリプションが作成されます。

ml_username は、Mobile Link サーバと同期することが許可されているユーザです。

同期ユーザ名の詳細については、「Mobile Link ユーザの概要」『Mobile Link - クライアント管理』を参照してください。

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。

異なるロケーションで指定されるオプションを dbmlsync が処理する方法については、「優先順位」『Mobile Link - クライアント管理』を参照してください。

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。

ネットワーク・プロトコルの詳細については、「CommunicationType (ctp) 拡張オプション」『Mobile Link - クライアント管理』を参照してください。

ADDRESS 句 Mobile Link サーバ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。

プロトコル・オプションの完全なリストについては、「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』を参照してください。

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。

異なるロケーションで指定されるオプションを dbmlsync が処理する方法については、「優先順位」『Mobile Link - クライアント管理』を参照してください。

オプションの完全なリストについては、「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』を参照してください。

備考

network-protocol、*protocol-options*、*options* は、複数の個所で設定できます。

異なるロケーションで指定されるオプションを dbmlsync が処理する方法については、「優先順位」『Mobile Link - クライアント管理』を参照してください。

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。

「dbmlsync 構文」『Mobile Link - クライアント管理』を参照してください。

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]」 332 ページ
- 「DROP SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 526 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 での同期の設計」『Mobile Link - クライアント管理』
- 「ISYSSYNC システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例では、Mobile Link ユーザ 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作成し、メモリを 3 MB に設定します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION memory='3m';
```

次の例では、FOR 句を省略し、パブリケーション sales_publication の設定を保存します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  ADDRESS 'host=test.internal;port=2439;
    security=ecc_tls'
  OPTION memory='2m';
```


CREATE SYNCHRONIZATION USER 文 [Mobile Link]

SQL Anywhere リモート・データベースでこの文を使用して、Mobile Link ユーザを作成します。

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

ml_username　Mobile Link ユーザを識別する名前。

Mobile Link ユーザの詳細については、「Mobile Link ユーザの概要」 『Mobile Link - クライアント管理』を参照してください。

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。

通信プロトコルの詳細については、「CommunicationType (ctp) 拡張オプション」 『Mobile Link - クライアント管理』を参照してください。

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 (TCPIP、TLS、HTTP、HTTPS) に応じて異なります。

プロトコル・オプションの完全なリストについては、「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』を参照してください。

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。

拡張オプションの詳細については、「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』を参照してください。

network-protocol、*protocol-options*、*options* は、複数の個所で設定できます。

異なるロケーションで指定されるオプションを dbmlsync が処理する方法については、「優先順位」『Mobile Link - クライアント管理』を参照してください。

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。

「dbmlsync 構文」『Mobile Link - クライアント管理』を参照してください。

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

- 「ALTER SYNCHRONIZATION USER 文 [Mobile Link]」 334 ページ
- 「DROP SYNCHRONIZATION USER 文 [Mobile Link]」 527 ページ
- 「Mobile Link クライアント／サーバ通信の暗号化」『SQL Anywhere サーバ - データベース管理』
- 「ISYSSYNC システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例は、TCP/IP 経由でサーバ・コンピュータ mlserver.mycompany.com と同期を行う、パスワード Sam を持つ Mobile Link ユーザ SSinger を作成します。ユーザ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

この文は、データベースに新しいテーブルを作成するために使用します。また、オプションでリモート・サーバ上にもテーブルを作成できます。

構文

CREATE [**GLOBAL TEMPORARY**] **TABLE** [*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*]
[*column-constraint* ⋯]

default-value :
 special-value
| *string*
| *global variable*
| [**-**] *number*
| **(** *constant-expression* **)**
| *built-in-function***(** *constant-expression* **)**
| **AUTOINCREMENT**
| **CURRENT DATABASE**
| **CURRENT REMOTE USER**
| **CURRENT UTC TIMESTAMP**
| **GLOBAL AUTOINCREMENT** [**(** *partition-size* **)**]
| **NULL**
| **TIMESTAMP**
| **UTC TIMESTAMP**
| **LAST USER**

special-value:
CURRENT {
  DATE
  | **TIME**
  | **TIMESTAMP**
  | **UTC TIMESTAMP**
  | **USER**
  | **PUBLISHER**
}
| **USER**

column-constraint :
[**CONSTRAINT** *constraint-name*] {


```
   UNIQUE[ CLUSTERED ]
  | PRIMARY KEY [ CLUSTERED ] [ ASC | DESC ]
  | REFERENCES table-name [ ( column-name ) ]
     [ MATCH [ UNIQUE ] { SIMPLE | FULL } ]
     [ actions ] [ CLUSTERED ]
   }
| [ CONSTRAINT constraint-name ] 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 ]
   [ ( column-name [ ASC | DESC ], ⋯ ) ]
   REFERENCES table-name
   [ ( column-name, ⋯ ) ]
   [ MATCH [ UNIQUE] { SIMPLE | FULL } ]
   [ actions ] [ CHECK ON COMMIT ] [ CLUSTERED ] [ FOR OLAP WORKLOAD ]

action :
ON { UPDATE | DELETE }
⋯{ 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

IN 句　テーブルが作成される DB 領域を指定します。テーブルが GLOBAL TEMPORARY テーブルの場合、IN 句は無視されます。

DB 領域の詳細については、「CREATE DBSPACE 文」 388 ページを参照してください。

CREATE TABLE 文を実行する前に、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。「default_dbspace オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

ENCRYPTED　ENCRYPTED 句は、テーブルを暗号化することを指定します。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。「テーブル暗号化を有効にする」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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　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。NOT TRANSACTIONAL 句の利点については、「テンポラリ・テーブルの編集」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

AT 句　*location-string* に指定された異なるサーバにリモート・テーブルを作成し、そのリモート・テーブルにマップするプロキシ・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。

たとえば、次の文は、テーブル a1 を TABLE Access ファイル *mydbfile.mdb* にマップします。

```
CREATE TABLE a1
AT 'access;d:¥mydbfile.mdb;;a1';
```

リモート・サーバについては、「CREATE SERVER 文」 444 ページを参照してください。プロキシ・テーブルについては、「CREATE EXISTING TABLE 文」 403 ページと「プロキシ・テーブルのロケーションの指定」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

Windows CE 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・サーバにサポートされている場合、そのデータベース・サーバに送信されます。

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。

テンポラリ・テーブルの特徴については、「テンポラリ・テーブルの編集」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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。「識別子」 7 ページを参照してください。
- **data-type**　カラムに格納されているデータ型。「SQL データ型」 47 ページを参照してください。
-

COMPRESSED　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 と PREFIX**

  INLINE 句と PREFIX 句は、BLOB (文字またはバイナリのデータ型のみ) を格納するときに使用されます。カラムに格納する最大 BLOB サイズをバイト単位で指定する場合は INLINE を使用します。INLINE 値を超える BLOB は、テーブル拡張ページのローの外部に格納されます。PREFIX 句は、BLOB のバイト数を指定して、ローを複製して格納するときに使用します。BLOB のプレフィクス・バイトのみを必要とする要求を処理する場合、プレフィクス・データのパフォーマンスを改善できます。

  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、文字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 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ければ、デフォルト値を使用することを強くおすすめします。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ました。たとえば、INLINE を大きな値に設定し、すべての BLOB をインラインで格納する場合、ローの処理パフォーマンスは低下することがあります。PREFIX の値を高くしすぎると、BLOB の格納に必要なディスク領域のサイズが増えます。これは、プレフィクス・データが BLOB の一部を複製したデータのためです。

  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。

  圧縮されたカラムの場合は、INLINE や PREFIX の設定に関係なく、INLINE と PREFIX の値に 0 が設定されたように動作します。したがって、プレフィクスは格納されず、BLOB はテーブル拡張ページに格納され、INDEX が指定されていれば (デフォルト設定)、BLOB のインデックスも作成されます。後でカラムを圧縮解除すると、INLINE と PREFIX で以前に有効だった設定がリストアされます。

- **[NO] INDEX**

BLOB を格納しているとき (文字データ型またはバイナリ・データ型のみ)、この句を使用して、BLOB インデックスを構築するかどうかを指定します。この句が指定されていない場合、データベース・サーバはインデックスを作成します。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、画像やマルチメディア・ファイルなど、BLOB 値の種類によっては、BLOB のインデックスは不要です。また、実際のところ、BLOB インデックスをオフにしてもパフォーマンスが改善することがあります。指定したカラムの 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 インデックスはデータベースのインデックスとは異なります。BLOB インデックスを作成すると、BLOB データへのランダム・アクセスが高速になります。一方、データベースのインデックスを作成すると、1 つ以上のカラムの値にインデックスが作成されます。

- **NOT NULL** NOTNULL が指定されているか、カラムが UNIQUE または PRIMARYKEY 制約下にある場合、カラムはいずれのローでも NULL を持つことはできません。
- **DEFAULT** *special-value* の詳細については、「特別値」 30 ページを参照してください。

  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。

  - **定数式** 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。

  - **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。

IDENTITY カラムは、AUTOINCREMENT デフォルトの使用に対する Transact-SQL 互換の代替手段です。SQL Anywhere では、IDENTITY カラムは AUTOINCREMENT デフォルトとして実装されています。詳細については、「特殊な IDENTITY カラム」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- **GLOBAL AUTOINCREMENT** このデフォルトは、SQL Remote レプリケーション環境または Mobile Link 同期環境で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 $np + 1$ ー $p(n + 1)$ 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 ー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。「イベントの概要」 『SQL Anywhere サーバ - データベース管理』を参照してください。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・タイプ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。

- **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。

  タイムスタンプ・カラムの詳細については、「Transact-SQL の特殊な timestamp カラムとデータ型」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

  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 [データベース] [Mobile Link クライアント]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

  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。「default_timestamp_increment オプション [データベース] [Mobile Link クライアント]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

  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。「グローバル変数」 38 ページを参照してください。

- **string**　「文字列」 8 ページを参照してください。
- **global-variable**　「グローバル変数」 38 ページを参照してください。

- **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。

  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。一般的に、カラムの制約はテーブルの 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 制約**　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化するかどうかを指定することもできます。CLUSTERED オプションとクラスタード・インデックスの詳細については、「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

  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。

- **外部キー**　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。

  指定した外部キー・カラムがテーブルに存在しない場合、カラムはプライマリ・テーブルの対応するカラムと同じデータ型で作成されます。このように自動的に作成されたカラムは、外部テーブルのプライマリ・キーの一部になることはできません。このため、同じテーブルのプライマリ・キーと外部キーの両方で使用されるカラムは、キーを作成する前に明示的に作成する必要があります。

  外部キー・カラム名とプライマリ・キーのカラム名とは、1 対 1 で対応する 2 つのリスト位置に従ってペアになります。FOREIGN KEY テーブル制約にプライマリ・テーブルのカラム名が指定されていない場合、プライマリ・キー・カラムが使用されます。外部キー・カラム名が指定されていない場合、外部キー・カラムにはプライマリ・テーブル内のカラムと同じ名前が付けられます。

  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。

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化するかどうを指定することもできます。CLUSTERED オプションとクラスタード・インデックスの詳細については、「クラスタード・インデックスの使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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、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 オプション** 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・テーブル中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。

- **MATCH 句**
  MATCH 句を使用すると、複数列の外部キーを使用するときに一致と考えられる内容を制御することができます。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。次に、指定できる一致タイプを示します。

  - **UNIQUE オプション** 参照元テーブルには、NOT NULL キー値と一致する値が 1 つだけあります (1 つ以上の NOT NULL カラム値があるキーは暗黙的にユニークです)。
  - **SIMPLE オプション** 参照テーブルのローに一致が発生するのは、キーの 1 つ以上のカラムが NULL の場合、またはすべてのカラム値が参照先テーブルのローにある対応するカラムと一致する場合です。
  - **FULL オプション** 参照テーブルのローに一致が発生するのは、キーの 1 つ以上のカラムが NULL の場合、またはすべてのカラム値が参照先テーブルのローにある対応するカラムと一致する場合です。
  - **SIMPLE UNIQUE オプション** 一致が発生するのは、SIMPLE と UNIQUE の両方について基準を満たしている場合です。
  - **FULL UNIQUE オプション** 一致が発生するのは、FULL と UNIQUE の両方について基準を満たしている場合です。

- **UNIQUE 制約**　*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あります。

  また、CLUSTERED キーワードを指定して、制約をクラスタ化するかどうを指定することもできます。CLUSTERED オプションの詳細については、「クラスタード・インデックスの使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

  ユニーク・インデックスの詳細については、「CREATE INDEX 文」 414 ページを参照してください。

- **CHECK 制約**　これで、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。

  TRUE、FALSE、UNKNOWN の各条件の詳細については、「NULL 値」 43 ページと「探索条件」 20 ページを参照してください。

- **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、式が評価されたときにデータベース・サーバによって変更されます。COMPUTE 式は、非決定的な値を返すことはできません。たとえば、CURRENT TIMESTAMP などの特別な値や非決定的関数を含め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を起動します。

- **CHECK ON COMMIT オプション** 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 オプションは、CASCADE、SET NULL、SET DEFAULT アクションを遅延しません。

  アクションを指定しないで CHECK ON COMMIT を使用すると、RESTRICT は、暗黙的に UPDATE と DELETE のアクションとみなされます。

- **FOR OLAP WORKLOAD オプション** 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改善することができます。特に、optimization_workload を OLAP に設定すると有効です。「optimization_workload オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

  OLAP の詳細については、「OLAP のサポート」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- **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 ー 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・テーブルとみなされます。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。「DECLARE LOCAL TEMPORARY TABLE 文」 495 ページを参照してください。

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。「allow_nulls_by_default オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

パーミッション

RESOURCE 権限が必要です。

別のユーザのテーブルを作成するには、DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE LOCAL TEMPORARY TABLE 文」 418 ページ
- 「ALTER TABLE 文」 336 ページ
- 「CREATE DBSPACE 文」 388 ページ
- 「CREATE EXISTING TABLE 文」 403 ページ
- 「DECLARE LOCAL TEMPORARY TABLE 文」 495 ページ
- 「DROP 文」 512 ページ
- 「特別値」 30 ページ
- 「SQL データ型」 47 ページ
- 「テーブルの作成」 『SQL Anywhere サーバ - SQL の使用法』
- 「allow_nulls_by_default オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「テンポラリ・テーブルの編集」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** コア機能。

  次はベンダ拡張です。

  - { IN | ON } *dbspace-name* 句
  - ON COMMIT 句
  - いくつかのデフォルト値

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、借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、リモート・サーバ SERVER_A にテーブル t1 を作成し、このリモート・テーブルにマップするプロキシ・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';
```

CREATE TRIGGER 文

この文は、テーブル内にトリガを作成するために使用します。

構文 1

CREATE 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* **)**]
BEGIN-statement

構文 2

CREATE 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* **)**]
BEGIN [**IF UPDATE (** *column-name* **) THEN**
[{ **AND** | **OR** } **UPDATE (** *column-name* **)**] ⋯]
BEGIN-statement
[**ELSEIF UPDATE (** *column-name* **) THEN**
[{ **AND** | **OR** } **UPDATE (** *column-name* **)**] ⋯]
BEGIN-statement
END IF]
END

column-list : *column-name* [, *column-name*, ...]

trigger-type : **BEFORE** | **AFTER** | **INSTEAD OF** | **RESOLVE**

trigger-event-list : *trigger-event* [, *trigger-event*, ...]

trigger-event : **DELETE** | **INSERT** | **UPDATE**

パラメータ

トリガ・イベント　トリガは次のイベントによって起動できます。以下に示す中から複数のトリガ・イベントを定義したり、UPDATE OF カラムリスト・イベントのいずれかを定義したりすることができます。

- **DELETE**　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**　このトリガ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**　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**　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・イベントは、*trigger-event-list* では使用できません。このトリガ用に定義されたトリガ・イベントであることが必要です。この句は、INSTEAD OF トリガ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。「IF 文」 580 ページを参照してください。

trigger-type　ロー・レベルのトリガを定義して、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。文レベルのトリガは、文の INSTEAD OF または AFTER の実行を定義できます。

INSTEAD OF トリガは、(非実体化) ビューに定義できるトリガの唯一の形式です。ビューで BEFORE または AFTER のトリガを定義しようとすると、SQLE_INVALID_TRIGGER_VIEW エラーが返されます。

INSTEAD OF トリガは、他の動作を、トリガ動作で置換します。INSTEAD OF トリガが起動すると、トリガ動作はスキップされ、代わりに指定された動作が実行されます。INSTEAD OF トリガは、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 トリガが起動すると、その文の結果としてロー・レベルのトリガが起動することはありません。ただし、文レベルのトリガの本文が、その他の処理を実行するため、結果として、その他のロー・レベルのトリガが実行されます。

INSTEAD OF トリガを定義する場合は、UPDATE OF *column-list* 句、ORDER 句、または WHEN 句は使用できません。

INSTEAD OF トリガの機能や制約の詳細については、「INSTEAD OF トリガ」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

BEFORE UPDATE トリガは、ローを対象に UPDATE が実行されるたびに起動されます。この場合、新しい値が古い値と異なるかどうかは問題ではありません。一方、AFTER UPDATE トリガは、新しい値が古い値と異なる場合にのみ起動されます。

RESOLVE トリガ型は、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。

FOR EACH 句　トリガをロー・レベルのトリガとして宣言するには、FOR EACH ROW 句を使用します。トリガを文レベルのトリガ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。わかりやすくするために、文レベルのトリガを宣言する場合、FOR EACH STATEMENT 句を入力することをおすすめします。

ORDER 句　同じタイミング (before、after、resolve) で起動される同じタイプ (insert、update、delete) のトリガは、ORDER 句を使用して起動する順序を決定します。ORDER 0 の指定は、ORDER 句を省略することと同じです。この句は、INSTEAD OF トリガでは使用できません。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。この句の性格上、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、トリガがロー・レベルのトリガと文レベルのトリガのどちらかによって異なります。ロー・レベルのトリガの場合、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 句を使うと、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 トリガの中で参照できます。REFERENCING NEW 句を使うと、BEFORE トリガ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。これらのテーブルのデフォルト名は deleted と inserted です。

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 RESOLVE UPDATE または RESOLVE UPDATE OF と一緒に使用してください。

WHEN 句　探索条件が真と評価されたローに対してだけ、トリガが起動されます。WHEN 句は、ロー・レベル・トリガと一緒にだけ使用できます。この句は、INSTEAD OF トリガでは使用できません。

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を作成し、データベースにトリガを格納します。

実体化ビュー (Materialized View) にトリガ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。

トリガをロー・レベルのトリガとして宣言すると、各ローを修正する前または後にトリガが実行されます。また、トリガを文レベルのトリガとして宣言すると、トリガ文全体が完了してから、トリガが実行されます。

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・ロックを設定するので、テーブルを排他的に使用してください。

関連する動作

オートコミット。

参照

- 「BEGIN 文」 356 ページ
- 「CREATE PROCEDURE 文」 423 ページ
- 「CREATE TRIGGER 文 [T-SQL]」 479 ページ
- 「DROP 文」 512 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「UPDATE 文」 728 ページ

標準と互換性

- **SQL/2003** 永続的ストアド・モジュール機能。いくつかの句はベンダ拡張です。

例

最初の例は、ロー・レベルのトリガを作成します。新しい部長が任命されたとき、その部の従業員の ManagerID カラムを更新する例を示します。

```
CREATE TRIGGER TR_change_managers
BEFORE UPDATE OF DepartmentHeadID
ON Departments
REFERENCING OLD AS old_dept NEW AS new_dept
FOR EACH ROW
BEGIN
   UPDATE Employees
   SET Employees.ManagerID=new_dept.DepartmentHeadID
   WHERE Employees.DepartmentID=old_dept.DepartmentID
END;
```

次の例はさらに複雑で、文レベルのトリガを扱います。最初に、テーブルを次のように作成します。

```
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を作成します。

```
CREATE TRIGGER insert-st 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;
```

次の例は、BEFORE UPDATE トリガで 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
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;
```

次の例は、BEFORE DELETE トリガで 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 トリガと UPDATE トリガで 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]

この文は、Adaptive Server Enterprise 互換の方法で、データベース内に新しいトリガを作成するために使用します。

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

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 トリガでは、Adaptive Server Enterprise の場合と同様に、テーブル名 deleted と inserted を使ってこれらのローにアクセスできます。Watcom-SQL CREATE TRIGGER 文では、これらのローには REFERENCING を使ってアクセスします。

トリガ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 トリガは、それを起動した文の後に実行されます。

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、DBA 権限が必要です。

CREATE TRIGGER はテーブルのすべてのローをロックするので、テーブルを排他的に使用し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」 473 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

CREATE VARIABLE 文

この文は、SQL 変数を作成するために使用します。

構文

CREATE VARIABLE *identifier data-type*

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。SET 文が別の値を割り当てるまで、この変数には 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。名前の解決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とき、または DROP VARIABLE 文を使うときに消去されます。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。

プロシージャとトリガのローカル変数は、複合文の中で宣言します (「複合文の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください)。

パーミッション

なし。

関連する動作

なし。

参照

- 「BEGIN 文」 356 ページ
- 「SQL データ型」 47 ページ
- 「DROP VARIABLE 文」 528 ページ
- 「SET 文」 677 ページ
- 「VAREXISTS 関数 [その他]」 280 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

この例は、データ型 VARCHAR(50) の変数 first_name を作成します。


```
CREATE VARIABLE first_name VARCHAR(50);
```

この例は、データ型 DATE の変数 birthday を作成します。

```
CREATE VARIABLE birthday DATE;
```

CREATE VIEW 文

この文は、データベース上にビューを作成するために使用します。ビューを使うと、格納されている方法とは異なる形でデータを参照できます。

構文

CREATE VIEW
[*owner.*]*view-name* [**(** *column-name*, ⋯ **)**]
AS *select-statement*
[**WITH CHECK OPTION**]

パラメータ

view-name　*view-name* は識別子です。デフォルトの所有者は現在のユーザ ID です。

column-name　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 (「SELECT 文」 669 ページを参照してください)。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。

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。SELECT 文に ORDER BY 句または GROUP BY 句を記述したり、UNION にすることはできます。ただし、SELECT 文に ORDER BY 句を指定すると、FIRST 句や TOP 句と一緒に使用する場合などは特に、ビュー定義の結果に影響を及ぼすことがあることに注意してください。

WITH CHECK OPTION 句　WITH CHECK OPTION 句は、SELECT 文で定義した探索条件に合わないビューには更新も挿入もできません。

備考

CREATE VIEW 文は、指定したビューを作成します。所有者を指定すると、別のユーザが所有するビューを作成できます。別のユーザのビューを作成するには、DBA 権限が必要です。

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。ただし、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、テーブルの所有者のユーザ ID を使って指定してください。別のユーザ ID ではテーブルを検索できなかったり、間違ったテーブルが取得される可能性があります。

ビューを定義する SELECT 文が GROUP BY 句と集合関数を含まないか、または UNION 文を伴わない場合は、ビューを更新できます。ビューを更新すると、元になっているテーブルも更新されます。

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として機能することから、「**パラメータ化されたビュー**」と呼ばれます。

パラメータ化されたビューは、等価の SELECT ブロックの本体を FROM 句内の抽出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として取るようなストアド・プロシージャに埋め込まれたクエリで、特に便利です。

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の実行時に SQL 変数が定義されていないと、カラムが見つからないという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。

関連する動作

オートコミット。

参照

- 「DROP 文」 512 ページ
- 「CREATE TABLE 文」 460 ページ
- 「CREATE MATERIALIZED VIEW 文」 420 ページ

標準と互換性

- **SQL/2003**　コア機能。ただし、パラメータ化されたビューはベンダ拡張です。

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門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、パラメータ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、次のパラメータ化されたビューでは、パラメータ 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、次の例はプロシージャ、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 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/2003**　ベンダ拡張。

DEALLOCATE DESCRIPTOR 文 [ESQL]

この文は、SQLDA に対応するメモリを解放するために使用します。

構文

DEALLOCATE DESCRIPTOR *descriptor-name*

descriptor-name : *string*

備考

データ項目、インジケータ変数、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]」 301 ページ
- 「SQLDA (SQL descriptor area)」 『SQL Anywhere サーバ - プログラミング』
- 「SET DESCRIPTOR 文 [ESQL]」 684 ページ

標準と互換性

- **SQL/2003** コア機能。

例

例については、「ALLOCATE DESCRIPTOR 文 [ESQL]」 301 ページを参照してください。

宣言セクション [ESQL]

この文は、Embedded SQL プログラムでホスト変数を宣言するために使用します。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。詳細については、「ホスト変数の使用」『SQL Anywhere サーバ - プログラミング』を参照してください。

パーミッション

なし。

参照

- 「BEGIN 文」 356 ページ

標準と互換性

- **SQL/2003** コア機能。

例

```
EXEC SQL BEGIN DECLARE SECTION;
char *surname, initials[5];
int dept;
EXEC SQL END DECLARE SECTION;
```

DECLARE 文

この文は、複合文 (BEGIN … END) 内の SQL 変数を宣言するために使用します。

構文

DECLARE *variable-name data-type*

備考

プロシージャ、トリガ、またはバッチの本体で使用される変数は、DECLARE 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。

Watcom-SQL プロシージャの本文またはトリガは複合文です。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 プロシージャまたはトリガには、そのような制約はありません。

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ
- 「DECLARE CURSOR 文 [T-SQL]」 494 ページ

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

例

次のバッチは、DECLARE 文の使用法を示し、メッセージを [サーバ・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
```

DECLARE CURSOR 文 [ESQL] [SP]

この文は、カーソルを宣言するために使用します。カーソルはクエリの結果を操作する主要な手段です。

構文 1 [ESQL]

DECLARE *cursor-name*
[**UNIQUE**]
[**NO SCROLL**
| **DYNAMIC SCROLL**
| **SCROLL**
| **INSENSITIVE**
| **SENSITIVE**
]
CURSOR FOR
{ *select-statement*
| *statement-name* [**FOR** { **UPDATE** [*cursor-concurrency*] | **READ ONLY** }]
| *call-statement* }

構文 2 [SP]

DECLARE *cursor-name*
[**NO SCROLL**
| **DYNAMIC SCROLL**
| **SCROLL**
| **INSENSITIVE**
| **SENSITIVE**
]
CURSOR
{ **FOR** *select-statement* [**FOR** { **UPDATE** [*cursor-concurrency*] | **READ ONLY** }]
| **FOR** *call-statement*
| **USING** *variable-name* }

cursor-name : *identifier*

statement-name : *identifier* | *hostvar*

variable-name : *identifier*

cursor-concurrency :
BY { **VALUES** | **TIMESTAMP** | **LOCK** }

パラメータ

UNIQUE 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、インジケータ変数の中に次の追加のオプションを設定します。

- **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 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。したがって、NO SCROLL カーソルを要求されると、SQL Anywhere は最も効率的な種類のカーソルである asensitive カーソルを提供します。「asensitive カーソル」『SQL Anywhere サーバ - プログラミング』を参照してください。

DYNAMIC SCROLL 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。「asensitive カーソル」『SQL Anywhere サーバ - プログラミング』を参照してください。

SCROLL 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」『SQL Anywhere サーバ - プログラミング』を参照してください。

SQL Anywhere は、結果セットのメンバ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 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 FETCHING は、他の INSERT、UPDATE、または DELETE 文の効果を参照しません。または別のカーソル上の他の PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT オペレーションを参照しません。INSENSITIVE カーソルからの FETCHING は、同じカーソル上の PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT オペレーションの効果を参照します。「insensitive カーソル」『SQL Anywhere サーバ - プログラミング』を参照してください。

SENSITIVE SENSITIVE として宣言されたカーソルは、結果セットのメンバシップまたは値の変更に依存します。「sensitive カーソル」『SQL Anywhere サーバ - プログラミング』を参照してください。

FOR statement-name 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。

FOR UPDATE | READ ONLY FOR READ ONLY として宣言されたカーソルは 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 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。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、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。

USING variable-name　ストアド・プロシージャ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。

- プロシージャへのパラメータ。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;
  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;
```

BY VALUES | TIMESTAMP | LOCK　Embedded SQL では、SELECT 文内またはカーソル宣言内に構文を含めることで、同時実行性の仕様を設定できます。ペシミスティックまたはオプティミスティックの同時実行性をカーソル・レベルに選択できます。このとき、DECLARE CURSOR 文または FOR 文を使用したオプション、または同時実行性を設定する特定のプログラミング・インタフェースの API が使用されます。文が更新可能で、カーソルが特定の同時制御メカニズムを指定しない場合、文の指定が使用されます。構文は次のとおりです。

- **FOR UPDATE BY LOCK**　データベース・サーバは、FETCH した結果セットのローに対して、意図的なロー・ロックをかけます。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 COMMIT または ROLLBACK されるまで保持されます。
- **FOR UPDATE BY { VALUES | TIMESTAMP }**　データベース・サーバは、keyset-driven 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ときにアプリケーションへ通知できるようにします。

備考

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、トリガ、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する必要があります。また、カーソル名をユニークにする必要があります。

カーソルが複合文内で宣言される場合、(Watcom-SQL または Transact-SQL 複合文での宣言にかかわらず) その複合文の間だけ存在します。

パーミッション

なし。

関連する動作

なし。

参照

- 「PREPARE 文 [ESQL]」 629 ページ
- 「OPEN 文 [ESQL] [SP]」 620 ページ
- 「EXPLAIN 文 [ESQL]」 541 ページ
- 「SELECT 文」 669 ページ
- 「CALL 文」 362 ページ
- 「FOR 文」 547 ページ

標準と互換性

- **SQL/2003**　コア機能。

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';
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement;
```

次の例は、ストアド・プロシージャ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 CURSOR 文 [T-SQL]

この文は、Adaptive Server Enterprise 互換の方法でカーソルを宣言するために使用します。

構文

DECLARE *cursor-name*
CURSOR FOR *select-statement*
[**FOR** { **READ ONLY** | **UPDATE** }]

cursor-name : *identifier*

select-statement : *string*

備考

Transact-SQL プロシージャの DECLARE CURSOR 文は、実行可能な文として扱われます。また、プロシージャのどこにでも指定できます。カーソルの宣言タスクは、文が実行されたときに有効になり、DEALLOCATE CURSOR 文が実行されるかプロシージャが完了するまで有効なままです。

SQL Anywhere では、カーソルが複合文内で宣言される場合、(Watcom-SQL または Transact-SQL 複合文での宣言にかかわらず) その複合文の間だけ存在します。

Transact-SQL のプロシージャ、トリガ、またはバッチの場合、DECLARE CURSOR 文はその他の実行可能文の後に指定できます。

パーミッション

なし。

関連する動作

なし。

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ

標準と互換性

- **SQL/2003** コア機能。FOR UPDATE と FOR READ ONLY オプションは、Transact-SQL 拡張。

DECLARE LOCAL TEMPORARY TABLE 文

この文は、ローカル・テンポラリ・テーブルを宣言するために使用します。

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

column-definition、*column-constraint*、*table-constraint*、*pctfree* の定義については、「CREATE TABLE 文」 460 ページを参照してください。

ON COMMIT　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　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

DECLARE LOCAL TEMPORARY TABLE 文はテンポラリ・テーブルを宣言します。

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 (「複合文の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください)。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。

プロシージャ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で作成するには、代わりに CREATE LOCAL TEMPORARY TABLE 文を使用します (「CREATE LOCAL TEMPORARY TABLE 文」 418 ページを参照してください)。

パーミッション

なし。

関連する動作

なし。

参照

- 「CREATE TABLE 文」 460 ページ
- 「CREATE LOCAL TEMPORARY TABLE 文」 418 ページ
- 「複合文の使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL／基本機能。

例

次の例は、ストアド・プロシージャ内のテンポラリ・テーブルの宣言方法を示します。

```
BEGIN
  DECLARE LOCAL TEMPORARY TABLE TempTab ( number INT );
  ...
END
```

DELETE 文

この文は、データベースからローを削除するために使用します。

構文

DELETE [**FIRST** | **TOP** *n*]
[**FROM**] [*owner*.]*table-name*
[**FROM** *table-list*]
[**WHERE** *search-condition*]
[**ORDER BY** { *expression* | *integer* } [**ASC** | **DESC**], ⋯]
[**OPTION(** *query-hint*, ... **)**]

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

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。

ビューを定義する SELECT 文が FROM 句の中でテーブルを 1 つだけ持ち、GROUP BY 句と集合関数を含まないか、または UNION 文を伴わない場合は、DELETE 文をビュー上で使用できます。

FIRST または TOP 句　この句は主に ORDER BY 句で使用します。この句によって、WHERE 句の条件を満たすローのサブセットだけを削除できます。TOP 値は、値が 0 以上の整数の定数または整数の変数にする必要があります。TOP の入力に変数は使用できません。

FROM 句　FROM 句は、ローの削除を開始するテーブルの位置を示します。DELETE 文に 第 2 FROM 句を指定すると、ジョインに基づいて指定したテーブルから削除するローが修飾されます。第 2 FROM 句がある場合、WHERE 句はこの第 2 FROM 句のローの条件を与えます。

第 2 FROM 句では、KEY ジョインや NATURAL ジョインなどの任意の複雑なテーブル式を使うことができます。FROM 句とジョインの詳細については、「FROM 句」 552 ページを参照してください。

次の文は、相関名を使用する 2 つの FROM 句を持つ DELETE 文の中のテーブル名に見られる潜在的なあいまいさを示しています。

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_2 AS alias_2
WHERE ...
```

テーブル table_1 は、第 1 FROM 句の中では相関名を使用しないで識別されますが、第 2 FROM 句の中では相関名を使用して識別されます。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には、テーブルのインスタンスは 2 つとする、一般規則の例外です。

次の例を考えます。

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

ここでは、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。

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 句のローの条件を与えます。

ORDER BY 句　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 句では序数のカラム番号を使用できません。

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、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 '*option-value*'**　MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザがクエリを処理するときに実体化ビュー (Materialized View) 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。「materialized_view_optimization オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **FORCE OPTIMIZATION**　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省略されます。ただし、コストベースの最適化を実行したい場合もあります。たとえば、クエリ処理時に実体化ビュー (Materialized View) 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、ビューのマッチング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、プロシージャ内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャに対するすべての呼び出しにオプティマイザ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。

単純なクエリとビューのマッチングに関する詳細については、「クエリ処理のフェーズ」『SQL Anywhere サーバ - SQL の使用法』と「実体化ビュー (Materialized View) によるパフォーマンスの向上」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- ***option-name = option-value*** 該当する文に対してのみ、パブリック・オプションやテンポラリ・オプションの設定よりも優先されるオプション設定を指定します。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 [互換性]」『SQL Anywhere サーバ - データベース管理』
  - 「max_query_tasks オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

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 TABLE 文」 718 ページ
- 「INSERT 文」 590 ページ
- 「INPUT 文 [Interactive SQL]」 585 ページ
- 「FROM 句」 552 ページ

標準と互換性

- **SQL/2003** コア機能。FROM 句での複数のテーブルの使用は、ベンダ拡張。

例

データベースから従業員 105 を削除します。

```
DELETE
FROM Employees
WHERE EmployeeID = 105;
```

FinancialData テーブルから、2000 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 );
```

DELETE (位置付け) 文 [ESQL] [SP]

この文は、カーソルの現在の位置でデータを削除するために使用します。

構文

DELETE [**FROM** *table-spec*] **WHERE CURRENT OF** *cursor-name*

cursor-name : *identifier* | *hostvar*

table-spec : [*owner*.]*correlation-name*

owner : *identifier*

備考

DELETE 文のこのフォームは、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、テーブル所有者が指定されない場合、*table-spec* がどの相関名に対しても最初に一致します。
  - 相関名がある場合、*table-spec* は相関名で識別されます。
  - 相関名がない場合、*table-spec* はカーソルのテーブル名として明確に識別可能にします。
- FROM 句が含まれ、テーブル所有者が指定されている場合、*table-spec* はカーソルのテーブル名として明確に指定できるようにします。
- 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」 728 ページ
- 「UPDATE (位置付け) 文 [ESQL] [SP]」 733 ページ
- 「INSERT 文」 590 ページ
- 「PUT 文 [ESQL]」 633 ページ

標準と互換性

- **SQL/2003**　コア機能。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拡張が含まれます。

例

次の文は、データベースから現在のローを削除します。

```
DELETE
WHERE CURRENT OF cur_employee;
```

DESCRIBE 文 [ESQL]

この文は、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するために使用します。

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*

cursor-name : *declared cursor*

sqlda-name : *identifier*

パラメータ

USER TYPES 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 DESCRIBE ALL を使用すると、データベース・サーバへの 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。INPUT 情報が最初に SQLDA に格納され、次に OUTPUT 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。

INPUT 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 パラメータを示し、DT_PROCEDURE_OUT は INOUT または OUT パラメータを示します。プロシージャ 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　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、プロシージャ中の各 INOUT または OUT パラメータに対して、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　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 のようにセットアップしてください。つまり、各カラムには、文字型のデータのエントリを 1 つ含めます。インジケータ変数がある場合、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　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・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。

SQLDA 構造体の使用の詳細については、「SQLDA (SQL descriptor area)」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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 と同等) または INPUT (BIND VARIABLES) 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 を割り当てます (「ALLOCATE DESCRIPTOR 文 [ESQL]」 301 ページを参照してください)。

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 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。「db_change_nchar_charset 関数」 『SQL Anywhere サーバ - プログラミング』を参照してください。

NCHAR データ型の記述方法については、データ型について説明されている「NCHAR データ型」 50 ページ、「NVARCHAR データ型」 52 ページ、「LONG NVARCHAR データ型」 49 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「ALLOCATE DESCRIPTOR 文 [ESQL]」 301 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ
- 「OPEN 文 [ESQL] [SP]」 620 ページ
- 「PREPARE 文 [ESQL]」 629 ページ

標準と互換性

- **SQL/2003**　コア機能。いくつかの句はベンダ拡張です。

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]

DESCRIBE 文を使用すると、テーブルやビューに格納されているすべてのカラム、テーブルのすべてのインデックス、およびストアド・プロシージャや関数で使用されているすべてのパラメータを取得できます。

構文

DESCRIBE [[**INDEX FOR**] **TABLE** | **PROCEDURE**] [*owner.*]*object-name*

パラメータ

INDEX FOR　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　記述するオブジェクトがテーブルまたはビューであることを示します。

PROCEDURE　記述するオブジェクトがプロシージャまたは関数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。出力は 4 つのカラムにフォーマットされます。

- **カラム**　記述するカラムの名前
- **タイプ**　カラムに格納されているデータ型
- **Null 入力可能**　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。出力は 4 つのカラムにフォーマットされます。

- **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。「クラスタード」、「統計情報」、「ハッシュド」、または「その他」を選択できます。

DESCRIBE PROCEDURE を使用すると、指定したプロシージャまたは関数が使用しているすべてのパラメータが表示されます。DESCRIBE PROCEDURE 文は、1 つのパラメータにつき 1 つのローを返します。出力は 3 つのカラムにフォーマットされます。

- **パラメータ**　パラメータの名前。
- **タイプ**　パラメータのユーザ型。
- **入力／出力**　プロシージャまたは関数に渡すデータまたは返されるデータに関する情報。

- 入力 - 呼び出し元はデータをプロシージャに渡しますが、返されるデータは受け取りません。
- 出力 - 呼び出し元はプロシージャまたは関数にデータを渡しませんが、返されるデータを受け取ります。
- 入力／出力 - 呼び出し元はプロシージャにデータを渡し、返されるデータを受け取ります。
- 結果 - プロシージャまたは関数は結果セットを返します。
- 戻り値 - プロシージャまたは関数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 はプロシージャまたは関数としてオブジェクトを記述しようとします。

パーミッション

なし。

関連する動作

なし。

参照

なし。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

Departments テーブル内のカラムを表示します。

```
DESCRIBE TABLE Departments;
```

この文の結果セットの例を示します。

カラム	型	Null 入力可能	プライマリ・キー
DepartmentID	integer	0	1
DepartmentName	char (40)	0	0
DepartmentHeadID	integer	0	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。

インデックス名	カラム	ユニーク	タイプ
IX_customer_name	Surname,GivenName	0	クラスタード

DETACH TRACING 文

診断トレーシング・セッションを終了するときに、この文を使用します。

構文

DETACH TRACING { WITH | WITHOUT } SAVE

パラメータ

WITH SAVE　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　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発行すると、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)。「sa_save_trace_data システム・プロシージャ」 946 ページを参照してください。

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。「sa_diagnostic_tracing_level テーブル」 775 ページを参照してください。

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

- 「ATTACH TRACING 文」 348 ページ
- 「REFRESH TRACING LEVEL 文」 642 ページ
- 「診断トレーシングを使用した詳細なアプリケーション・プロファイリング」『SQL Anywhere サーバ - SQL の使用法』
- 「sa_diagnostic_tracing_level テーブル」 775 ページ
- 「sa_save_trace_data システム・プロシージャ」 946 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

DISCONNECT 文 [ESQL] [Interactive SQL]

この文は、データベースとの現在の接続を切断するために使用します。

構文

DISCONNECT [*connection-name* | **CURRENT** | **ALL**]

connection-name : *identifier*, *string*, or *hostvar*

備考

DISCONNECT 文はデータベース・サーバとの接続を切断し、データベース・サーバ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すべてが切断されます。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。

現在の接続以外の接続を削除する方法については、「DROP CONNECTION 文」 515 ページを参照してください。

この文は、プロシージャ、トリガ、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

- 「CONNECT 文 [ESQL] [Interactive SQL]」 375 ページ
- 「SET CONNECTION statement [Interactive SQL] [ESQL]」 683 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL／基本機能。

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;
```


DROP 文

この文は、データベースからオブジェクトを削除するために使用します。

構文

DROP
{ **DOMAIN** | **DATATYPE** } *datatype-name*
| **DBSPACE** *dbspace-name*
| **EVENT** *event-name*
| **FUNCTION** [*owner.*]*function-name*
| **INDEX** { [[*owner.*]*table-name.*]*index-name* | [[*owner.*]*materialized-view-name.*]*index-name* }
| **MESSAGE** *msgnum*
| **PROCEDURE** [*owner.*]*procedure-name*
| **TABLE** [*owner.*]*table-name*
| **TRIGGER** [[*owner.*]*table-name.*]*trigger-name*
| **VIEW** [*owner.*]*view-name*
| **MATERIALIZED VIEW** [*owner.*]*materialized-view-name*

備考

DROP 文は、上記のデータベース・オブジェクトの定義を削除します。オブジェクトが DB 領域である場合、DB 領域内のすべてのテーブルを削除してから、DB 領域を削除します。オブジェクトがテーブルまたは実体化ビュー (Materialized View) である場合、テーブルの中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。また、テーブルまたは実体化ビュー (Materialized View) のすべてのインデックスとキーも削除されます。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。「事前定義の DB 領域」 『SQL Anywhere サーバ - データベース管理』を参照してください。

別の接続から使用されているオブジェクトに文が影響を及ぼす場合、DROP TABLE、DROP MATERIALIZED VIEW、DROP INDEX、DROP DBSPACE、DROP PROCEDURE、DROP FUNCTION は許可されません。テーブルに依存する実体化ビュー (Materialized View) がある場合、DROP TABLE は許可されません。

DROP TABLE、DROP MATERIALIZED VIEW、DROP VIEW によって、依存しているすべての非実体化ビュー (Non-materialized View) の状態は INVALID になります。テーブル、ビュー、実体化ビュー (Materialized View) 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・プロシージャを使用します。「sa_dependent_views システム・プロシージャ」 890 ページを参照してください。

データ型をテーブル・カラム、プロシージャの引数、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 ANSI/ISO SQL3 の草案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、または DBA 権限を持つユーザはすべて、DROP 文を実行できます。

DROP DBSPACE の場合は、データベースへの唯一の接続でなければなりません。

テーブルに対する ALTER パーミッションを持つユーザは、DROP TRIGGER を実行できます。

テーブルに対する REFERENCES パーミッションを持つユーザは、DROP INDEX 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が切断されるまで削除できません。

データベースでスナップショット・アイソレーションを有効にしている場合、スナップショット・トランザクション内では DROP INDEX 文を使用できません。「スナップショット・アイソレーション」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

関連する動作

オートコミット。Interactive SQL の [結果] ウィンドウ枠の [結果] タブをクリアします。DROP TABLE、DROP VIEW、DROP MATERIALIZED VIEW、DROP INDEX は、現在の接続しているすべてのカーソルを閉じます。

DROP TABLE は、ローカル・テンポラリ・テーブルを削除するときに使用できますが、DROP INDEX は、ローカル・テンポラリ・テーブル上のインデックスを削除するときに使用できません。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。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。

ビューを削除すると、すべてのプロシージャとトリガ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やトリガは、そのビューが存在しないことを反映します。ビューを頻繁に削除、作成すると、プロシージャやトリガ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 DATABASE 文」 379 ページ
- 「CREATE DOMAIN 文」 392 ページ
- 「CREATE INDEX 文」 414 ページ
- 「CREATE FUNCTION 文」 408 ページ
- 「CREATE PROCEDURE 文」 423 ページ
- 「CREATE TABLE 文」 460 ページ
- 「CREATE TRIGGER 文」 473 ページ
- 「CREATE VIEW 文」 482 ページ
- 「CREATE STATISTICS 文」 452 ページ
- 「CREATE MATERIALIZED VIEW 文」 420 ページ

標準と互換性

- **SQL/2003**　コア機能。ただし、実体化ビュー (Materialized View) のサポートはベンダ拡張です。

例

データベースから Department テーブルを削除します。

```
DROP TABLE Departments;
```


データベースから EmployeesAndDepartments ビューを削除します。

```
DROP VIEW EmployeesAndDepartments;
```

ProductIDsPerCustomer 実体化ビュー (Materialized View) から price インデックスを削除します。

```
DROP INDEX ProductIDsPerCustomer.price;
```

DROP CONNECTION 文

この文は、データベースとのユーザの接続を切断するために使用します。

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を切断します。

connection-id パラメータは整数の定数です。sa_conn_info システム・プロシージャを使用して、*connection-id* を取得できます。

この文は、プロシージャ、トリガ、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 権限が必要です。

関連する動作

なし。

参照

- 「CONNECT 文 [ESQL] [Interactive SQL]」 375 ページ
- 「sa_conn_info システム・プロシージャ」 880 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次のプロシージャは、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

この文は、データベースに関連するすべてのデータベース・ファイルを削除するために使用します。

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、トリガ、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 CE 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」 379 ページ
- 「DatabaseKey 接続パラメータ [DBKEY]」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

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 EXTERNLOGIN 文

この文は、SQL Anywhere カタログから外部ログインを削除するために使用します。

構文

DROP EXTERNLOGIN *login-name* **TO** *remote-server*

パラメータ

DROP 句　ローカル・ユーザ・ログイン名を指定します。

TO 句　リモート・サーバ名を指定します。このサーバのローカル・ユーザ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 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE EXTERNLOGIN 文」 406 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

```
DROP EXTERNLOGIN DBA TO sybase1;
```

DROP PUBLICATION 文 [Mobile Link] [SQL Remote]

この文は、パブリケーションを削除するときに使用します。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

DROP PUBLICATION [*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。

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。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]」 321 ページ
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 436 ページ
- SQL Anywhere Mobile Link クライアント: 「データのパブリッシュ」 『Mobile Link - クライアント管理』
- Ultra Light Mobile Link クライアント :「Ultra Light DROP PUBLICATION 文」 『Ultra Light - データベース管理とリファレンス』

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

この文は、データベースからのメッセージ・タイプ定義の削除に使用します。

構文

DROP REMOTE MESSAGE TYPE *message-system*

message-system: **FILE** | **FTP** | **MAPI** | **SMTP** | **VIM**

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。

> **注意**
> VIM および MAPI のサポートは廃止されました。

パーミッション

DBA 権限が必要です。指定されたメッセージ・タイプに REMOTE または CONSOLIDATE パーミッションを持つユーザがいると、そのタイプは削除できません。

関連する動作

オートコミット。

参照

- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」 440 ページ
- 「メッセージ・タイプの使用」 『SQL Remote』.

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 SERVER 文

この文は、SQL Anywhere カタログからリモート・サーバを削除するために使用します。

構文

DROP SERVER *server-name*

備考

DROP SERVER は、SQL Anywhere カタログからリモート・サーバを削除します。この文を成功させるには、リモート・サーバ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

ユーザ DBA だけがリモート・サーバを削除できます。

Windows CE 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」 444 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

```
DROP SERVER ase_prod;
```

DROP SERVICE 文

この文を使用して 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 権限が必要です。

関連する動作

なし。

参照

- 「ALTER SERVICE 文」 327 ページ
- 「CREATE SERVICE 文」 448 ページ
- 「ISYSWEBSERVICE システム・テーブル」 760 ページ

例

tables という名前の Web サービスを削除するには、次の文を実行します。

```
DROP SERVICE tables;
```

DROP STATEMENT 文 [ESQL]

この文は、文のリソースを解放するために使用します。

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]」 629 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次は、DROP STATEMENT の使用例です。

```
EXEC SQL DROP STATEMENT S1;
EXEC SQL DROP STATEMENT :stmt;
```


DROP STATISTICS 文

この文は、指定したカラムに関するカラムの統計情報をすべて削除するために使用します。

構文

DROP STATISTICS [**ON**] [*owner.*]*table-name* [**(** *column-list* **)**]

備考

SQL Anywhere オプティマイザは、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、オプティマイザは必要な統計情報にアクセスできなくなります。統計情報にアクセスできない場合、オプティマイザはきわめて非効率的なデータ・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 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE STATISTICS 文」 452 ページ
- 「ISYSCOLSTAT システム・テーブル」 753 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

DROP SUBSCRIPTION 文 [SQL Remote]

この文は、パブリケーションからユーザのサブスクリプションを削除するのに使用します。

構文

DROP SUBSCRIPTION TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*, ⋯

subscription-value: *string*

subscriber-id: *string*

パラメータ

publication-name　このパブリケーション名でユーザ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のユーザ 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 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 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 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE SUBSCRIPTION 文 [SQL Remote]」 453 ページ
- 「ISYSSUBSCRIPTION システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、pub_contact publication に対するユーザ 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 SUBSCRIPTION 文 [Mobile Link]

この文を使用して、Mobile Link リモート・データベースから同期サブスクリプションを削除します。

構文

DROP SYNCHRONIZATION SUBSCRIPTION
TO *publication-name*
[**FOR** *ml_username*, …]

パラメータ

TO 句 パブリケーション名を指定します。

FOR 句 1 人以上の Mobile Link ユーザを指定します。

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]」 332 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 455 ページ
- 「ISYSSYNC システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例では、Mobile Link ユーザ 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;
```

DROP SYNCHRONIZATION USER 文 [Mobile Link]

この文を使用して、SQL Anywhere リモート・データベースから 1 つ以上の同期ユーザを削除します。

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を削除します。

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

ユーザ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]」 334 ページ
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」 458 ページ
- 「ISYSSYNC システム・テーブル」 759 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

データベースから Mobile Link ユーザ ml_user1 を削除します。

```
DROP SYNCHRONIZATION USER ml_user1;
```


DROP VARIABLE 文

この文は、SQL 変数を削除するために使用します。

構文

DROP VARIABLE *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。

パーミッション

なし。

関連する動作

なし。

参照

- 「CREATE VARIABLE 文」 480 ページ
- 「SET 文」 677 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

EXCEPT 文

2 つ以上のクエリの結果セット間の差を計算します。

構文

[**WITH** *temporary-views*] *query-block*
EXCEPT [**ALL** | **DISTINCT**] *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*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

パラメータ

注意
query-block で FOR、FOR XML、WITH または OPTION の句を使用することはできません。

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 '*option-value*'** MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザがクエリを処理するときに実体化ビュー (Materialized View) 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。「materialized_view_optimization オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **FORCE OPTIMIZATION** 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省略されます。ただし、コストベースの最適化を実行したい場合もあります。たとえば、クエリ処理時に実体化ビュー (Materialized View) 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、ビューのマッチング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、プロシージャ内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャに対するすべての呼び出しにオプティマイザ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。

単純なクエリとビューのマッチングに関する詳細については、「クエリ処理のフェーズ」『SQL Anywhere サーバ - SQL の使用法』と「実体化ビュー (Materialized View) によるパフォーマンスの向上」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- ***option-name = option-value*** 該当する文に対してのみ、パブリック・オプションやテンポラリ・オプションの設定よりも優先されるオプション設定を指定します。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 [互換性]」『SQL Anywhere サーバ - データベース管理』
  - 「max_query_tasks オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

備考

複数のクエリ・ブロックの結果セット間の差は、EXCEPT または EXCEPT ALL を使用して 1 つの結果として取得できます。EXCEPT DISTINCT は EXCEPT と同じです。

query-block では、それぞれ select リストの中の項目数が同じになるようにしてください。

EXCEPT ALL の結果セットにあるローの数は、別々のクエリの結果セットにあるローの数間の差に一致します。

EXCEPT の結果は EXCEPT ALL と同じです。ただし、EXCEPT を使用する場合の異なる点は、結果セット間の差が計算される前に重複するローが削除されることです。

2 つの select リスト中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。UNION の最初のクエリ指定は、ORDER BY 句と一致する名前を判断するために使用します。

表示されるカラム名は、最初の *query-block* に対して表示されるカラム名と同じです。結果セットのカラム名をカスタマイズするには、*query-block* で WITH 句を使用するという方法もあります。

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」 597 ページ
- 「UNION 文」 720 ページ

標準と互換性

- **SQL/2003** EXCEPT DISTINCT はコア機能です。EXCEPT ALL は F304 です。

例

EXCEPT の使用例については、「集合演算子と NULL」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

EXECUTE 文 [ESQL]

この文は、準備された SQL 文を実行するために使用します。

構文 1

EXECUTE *statement*
[**USING** { *hostvar-list* | **DESCRIPTOR** *sqlda-name* }]
[**INTO** { *into-hostvar-list* | **DESCRIPTOR** *into-sqlda-name* }]
[**ARRAY :***integer*]

statement : *identifier* | *hostvar* | *string*

sqlda-name : *identifier*

into-sqlda-name : *identifier*

構文 2

EXECUTE IMMEDIATE *statement*

statement : *string* | *hostvar*

パラメータ

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) 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 (「ESQL でのカーソルの使用」 『SQL Anywhere サーバ - プログラミング』を参照してください)。

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 (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]」 536 ページ
- 「PREPARE 文 [ESQL]」 629 ページ
- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない機能。

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を呼び出すために使用します。構文 2 は、Transact-SQL で準備された SQL 文を実行するために使用します。

構文 1

EXECUTE [*@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 はストアド・プロシージャを実行します。オプションとしてプロシージャ・パラメータを指定し、出力された値を取得し、ステータス情報を返します。

EXECUTE 文は、Transact-SQL 互換性のために用意されていますが、Transact-SQL または Watcom-SQL のバッチとプロシージャにも使えます。

構文 2 を使用すると、Transact-SQL のストアド・プロシージャとトリガの中で文を実行できます。EXECUTE 文は、プロシージャとトリガ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、プロシージャに渡されるパラメータ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。

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 THEN
    SELECT 1 AS a
```

Transact-SQL のストアド・プロシージャとトリガの中で文を実行することもできます。「EXECUTE IMMEDIATE 文 [SP]」 536 ページを参照してください。

パーミッション

プロシージャの所有者であるか、そのプロシージャに対する 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」 362 ページ
- 「EXECUTE 文 [ESQL]」 532 ページ

- 「EXECUTE IMMEDIATE 文 [SP]」 536 ページ

例

次のプロシージャは、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、パラメータに 23 の入力値を提供するプロシージャを実行します。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 [サーバ・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を実行する代替方法です。これは、パラメータがいくつかあるときに便利です。

```
EXECUTE p1 @var = 23;
```

次の文は、パラメータにデフォルト値を使用するプロシージャを実行します。

```
EXECUTE p1;
```

次の文は、プロシージャを実行し、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;
```


EXECUTE IMMEDIATE 文 [SP]

この文は、動的に作成された文をプロシージャ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 RESULT SET** { **ON** | **OFF** }

構文 2

EXECUTE (*string-expression* **)**

パラメータ

WITH QUOTES　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みなされます。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が呼び出される場合があります。「quoted_identifier オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

WITH ESCAPES　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
```

WITH RESULT SET　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が結果セット

を返すように指定されます。この句を含めない場合、文が結果セットを生成する場合にプロシージャが呼び出されると、エラーがレポートされます。

注意
デフォルト・オプションは WITH RESULT SET OFF であり、文の実行時に結果セットが生成されないことを意味します。

備考

EXECUTE 文は、プロシージャとトリガ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、プロシージャに渡されるパラメータ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 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・プロシージャとトリガでは、構文 2 しか使用できません。

パーミッション

なし。文は、プロシージャの所有者のパーミッションを使って実行されます。プロシージャを呼び出すユーザのパーミッションは使いません。

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。

プロシージャにおける EXECUTE IMMEDIATE 文の使用の詳細については、「プロシージャでの EXECUTE IMMEDIATE 文の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

参照

- 「CREATE PROCEDURE 文」 423 ページ
- 「BEGIN 文」 356 ページ
- 「EXECUTE 文 [ESQL]」 532 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL／基本機能。

例

次の文は、テーブル名がパラメータとしてプロシージャに提供されているテーブルを作成します。EXECUTE IMMEDIATE 文はすべて 1 行にしてください。

```
CREATE PROCEDURE CreateTableProc(
                IN tablename char(30)
                )
BEGIN
  EXECUTE IMMEDIATE
  'CREATE TABLE ' || tablename ||
  ' ( column1 INT PRIMARY KEY)'
END;
```


プロシージャを呼び出して、テーブル mytable を作成します。

```
CALL CreateTableProc( 'mytable' );
```

結果セットを返すクエリを使用する EXECUTE IMMEDIATE の例については、「プロシージャでの EXECUTE IMMEDIATE 文の使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

EXIT 文 [Interactive SQL]

この文は、Interactive SQL を終了するために使用します。

構文

{ **EXIT** | **QUIT** | **BYE** } [*return-code*]

return-code: *number* | *connection-variable*

備考

この文は、Interactive SQL を Windows プログラムとして実行する場合、またはコマンド・プロンプト (バッチ) ・モードで実行しているときに Interactive SQL を終了する場合、Interactive SQL ウィンドウを閉じます。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・ファイルで使用すると、Interactive SQL コマンド・ファイルにコマンドの成功または失敗が示されます。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。

参照

- 「SET OPTION 文」 686 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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;
```

次の Windows バッチ・ファイルは、コマンド・プロンプト上に Error = 1 と出力します。

```
dbisql -c "DSN=SQL Anywhere 10 Demo" EXIT 1
IF ERRORLEVEL 1 ECHO "Errorlevel is 1"
```

EXPLAIN 文 [ESQL]

この文は、特定のカーソルに対して使う最適化方法のテキスト仕様を取得するために使用します。

構文

EXPLAIN PLAN FOR CURSOR *cursor-name*
{ **INTO** *hostvar* | **USING DESCRIPTOR** *sqlda-name* }

cursor-name : *identifier* or *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改善」 『SQL Anywhere サーバ - 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]」 489 ページ
- 「PREPARE 文 [ESQL]」 629 ページ
- 「FETCH 文 [ESQL] [SP]」 543 ページ

- 「CLOSE 文 [ESQL] [SP]」 368 ページ
- 「OPEN 文 [ESQL] [SP]」 620 ページ
- 「ESQL でのカーソルの使用」 『SQL Anywhere サーバ - プログラミング』
- 「SQLCA (SQL Communication Area)」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

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;
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]

この文は、カーソルを再配置し、そこからデータを取り出すために使用します。

構文

FETCH *cursor-position cursor-name*
[**INTO** { *hostvar-list* | *variable-list* } | **USING DESCRIPTOR** *sqlda-name*]
[**PURGE**]
[**BLOCK** *n*]
[**FOR UPDATE**]
[**ARRAY** *fetch-count*]
INTO *variable-list* [**FOR UPDATE**]

cursor-position :
NEXT | **PRIOR** | **FIRST** | **LAST**
| { **ABSOLUTE** | **RELATIVE** } *row-count*

row-count : *number* or *hostvar*

cursor-name : *identifier* or *hostvar*

hostvar-list : may contain indicator variables

variable-list : stored procedure variables

sqlda-name : *identifier*

fetch-count : *integer* or *hostvar*

パラメータ

INTO 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。FETCH 文に配置パラメータがあり、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** 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** 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** 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。したがって、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** ABSOLUTE 配置パラメータを使用すると、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(「プロシージャとトリガでのカーソルの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください)。

  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** ABSOLUTE 1 の省略形。
- **LAST** ABSOLUTE -1 の省略形。

カーソル位置の問題
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、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、テンポラリ・テーブルを作成してカーソルを開いた場合に起こります (詳細については、「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 - SQL の使用法』を参照してください)。
UPDATE 文によって、カーソル中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に格納します。後に続くフェッチは、データベース・サーバへ新しい要求を行わないで、これらのバッファ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 という特別な値は、要求をデータベース・サーバに送信し、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 接続パラメータを使用します。

BLOCK 句を指定しない場合は、OPEN に指定された値が使われます。「OPEN 文 [ESQL] [SP]」 620 ページを参照してください。

FETCH RELATIVE 0 は常にローを再フェッチします。

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。「ロックの仕組み」『SQL Anywhere サーバ - SQL の使用法』と、「SELECT 文」669 ページの FOR UPDATE 句を参照してください。

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。

ワイド・フェッチの詳細な使用例については、「一度に複数のローをフェッチする」『SQL Anywhere サーバ - プログラミング』を参照してください。

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。エラーではなくて SQLCOUNT が 0 の場合、有効なローが 1 つフェッチされたことを示します。

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-list* 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

なし。

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ
- 「PREPARE 文 [ESQL]」 629 ページ
- 「OPEN 文 [ESQL] [SP]」 620 ページ
- 「ESQL でのカーソルの使用」 『SQL Anywhere サーバ - プログラミング』
- 「プロシージャとトリガでのカーソルの使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「FOR 文」 547 ページ
- 複数の結果セットを取得するには、「RESUME 文」 652 ページ

標準と互換性

- **SQL/2003** コア機能。プロシージャでの使用は、永続的ストアド・モジュール機能です。

例

次は、Embedded SQL の例です。

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT EmployeeID, Surname FROM Employees;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

次は、プロシージャの例です。

```
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

この文は、カーソル内の各ローに対して一度だけ、文の実行を繰り返すために使用します。

構文

[*statement-label* :]
FOR *for-loop-name* **AS** *cursor-name* [*cursor-type*] **CURSOR**
{ **FOR** *statement* [**FOR** { **UPDATE** *cursor-concurrency* | **FOR 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

NO SCROLL 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。したがって、NO SCROLL カーソルを要求されると、SQL Anywhere は最も効率的な種類のカーソルである asensitive カーソルを提供します。「asensitive カーソル」 『SQL Anywhere サーバ - プログラミング』を参照してください。

DYNAMIC SCROLL 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。「asensitive カーソル」 『SQL Anywhere サーバ - プログラミング』を参照してください。

SCROLL 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」 『SQL Anywhere サーバ - プログラミング』を参照してください。

SQL Anywhere は、結果セットのメンバ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 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 FETCHING は、他の INSERT、UPDATE、または DELETE 文の効果を参照しません。または別のカーソル上の他の PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT オペレーションを参照しません。INSENSITIVE カーソルからの FETCHING は、同じカーソル上の PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT オペレーションの効果を参照します。「insensitive カーソル」『SQL Anywhere サーバ - プログラミング』を参照してください。

SENSITIVE SENSITIVE として宣言されたカーソルは、結果セットのメンバシップまたは値の変更に依存します。「sensitive カーソル」『SQL Anywhere サーバ - プログラミング』を参照してください。

FOR UPDATE | READ ONLY FOR READ ONLY として宣言されたカーソルは 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 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。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、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。

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。

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 リスト中の式のデータ型です。名前は、それらが存在する select リスト項目のエイリアスです。そうでない場合、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。

パーミッション

なし。

関連する動作

なし。

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ
- 「FETCH 文 [ESQL] [SP]」 543 ページ
- 「CONTINUE 文 [T-SQL]」 378 ページ
- 「LOOP 文」 614 ページ

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

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
```

FORWARD TO 文

この文は、ネイティブ SQL 文をリモート・サーバへ送信するために使用します。

構文 1

FORWARD TO *server-name sql-statement*

構文 2

FORWARD TO [*server-name*]

備考

FORWARD TO 文は、ユーザがパススルー接続が必要なサーバ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 を発行してください。

  パススルー・モードでリモート・サーバからエラーが返された場合は、FORWARD TO 文を発行してパススルー・モードをオフにしてください。

ユーザのために 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に返されます。

文が要求されたサーバに渡されると、結果はすべて、クライアント・プログラムで認識できるフォームに変換されます。

server-name　リモート・サーバの名前。

SQL-statement　リモート・サーバのネイティブ 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。

> **注意**
> FORWARD TO 文はサーバ・ディレクティブであり、ストアド・プロシージャ、トリガ、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。

例

次の例では、SQL 文をリモート・サーバ RemoteASE に送信します。

```
FORWARD TO RemoteASE { SELECT * FROM titles }
```

次の例は、リモート・サーバ 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/2003**　ベンダ拡張。

FROM 句

この句は、SELECT 文、UPDATE 文、または DELETE 文に必要なデータベース・テーブルまたはビューを指定するために使用します。

構文

FROM *table-expression*, ⋯

table-expression:
table-name
| *view-name*
| *procedure-name*
| *derived-table-name*
| *lateral-derived-table-name*
| *joined-table-name*
| **(** *table-expression*, ⋯ **)**

table-name :
[*userid***.**]*table-name*
[[**AS**] *correlation-name*]
[**WITH (** *table-hint* | **NO INDEX** | **INDEX (** *index-name* **))** | **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-name :
(*select-statement* **)**
[**AS**] *correlation-name* [**(** *column-name*, ⋯ **)**]

lateral-derived-table-name :
LATERAL (*select-statement* | *table-expression* **)**
[**AS**] *correlation-name* [**(** *column-name*, ⋯ **)**]

joined-table-name :
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**]

table-hint:
HOLDLOCK
| **NOLOCK**
| **READCOMMITTED**
| **READPAST**
| **READUNCOMMITTED**
| **REPEATABLEREAD**
| **SERIALIZABLE**
| **UPDLOCK**
| **XLOCK**
| **FASTFIRSTROW**

パラメータ

table-name　ベース・テーブルまたはテンポラリ・テーブル。ユーザ ID を指定すると、他のユーザが所有するテーブルを修正できます。ユーザ ID を指定しないと、現在のユーザ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 (「グループが所有するテーブルの参照」 『SQL Anywhere サーバ - データベース管理』を参照してください)。

WITH (INDEX (*index-name*)) 句と、同等の FORCE INDEX (*index-name*) 句は、テーブルのインデックス・ヒントを指定します。この句はクエリ・オプティマイザ・プラン選択アルゴリズムを無効にし、使用可能な他のアクセス・プランに関係なく、最適化されたクエリに対して、指定されたインデックスを使用してテーブルにアクセスするよう要求します。相関名 1 つにつき指定できるインデックス・ヒントは、1 つだけです。インデックス・ヒントは、ベース・テーブルとテンポラリ・テーブルでのみ指定できます。

WITH (NO INDEX) 句は、テーブルの連続スキャンを強制的に実行します。たとえば、次の SELECT 文は、Customers テーブルからの選択を強制的に連続して実行します。

```
SELECT * FROM Customers
  WITH ( NO INDEX )
  WHERE Customers.ID >= 500
  ORDER BY Customers.ID DESC;
```

高度な機能
インデックス・ヒントはクエリ・オプティマイザを上書きするため、経験のあるユーザのみ使用してください。インデックス・ヒントを使用すると、次善のアクセス・プランが使用され、パフォーマンスが低下することがあります。

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 ID を指定して、他のユーザが所有するビューを修飾できます。ユーザ ID を指定しないと、現在のユーザ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・プロシージャです。プロシージャは、SELECT 文の FROM 句のみで使用できます。UPDATE 文または DELETE 文では使用できません。プロシージャにパラメータを指定しない場合でも、プロシージャ名の後のカッコは必要です。ストアド・プロシージャ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;
```

「ProcCall アルゴリズム」 『SQL Anywhere サーバ - SQL の使用法』と「プロシージャ統計」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

derived-table-name　FROM 句内のテーブルまたはビュー名の代わりに SELECT 文を指定できます。これで、ビューを作成しないで、グループ上でグループ、またはグループでジョインが使用できます。この方法で作成したテーブルが、抽出テーブルです。

lateral-derived-table-name　外部参照を含む抽出テーブル、ストアド・プロシージャ、ジョインされたテーブル。FROM 句の外部参照を使用する場合は、ラテラル抽出テーブルを使用します。外部参照についての詳細は、「外部参照」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

外部参照を使用できるのは、FROM 句のラテラル抽出テーブルの前のテーブルに対してだけです。たとえば、*select-list* 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 (外部参照が強調表示されています) は有効です。

```
SELECT *
 FROM A, LATERAL( B LEFT OUTER JOIN C ON ( A.x = B.x ) ) LDT;

SELECT *
 FROM A, LATERAL( SELECT * FROM B WHERE A.x = B.x ) LDT;

SELECT *
 FROM A, LATERAL( procedure-name( A.x ) ) LDT;
```

LATERAL (*table-expression*) の指定は、LATERAL (SELECT * FROM *table-expression*) の指定と同等です。

correlation-name-name　文の他の部分のオブジェクトを参照するときに使用する識別子。

同じ相関名をテーブル式の同じテーブルに対して 2 回使用する場合、そのテーブルは、一度だけリストされたものとして扱われます。たとえば、次の 2 つの SELECT 文は同じです。

```
SELECT *
 FROM SalesOrders
 KEY JOIN SalesOrderItems, SalesOrders
  KEY JOIN Employees;

SELECT *
  FROM SalesOrders
  KEY JOIN SalesOrderItems
  KEY JOIN Employees;
```

一方、次の文では Person テーブルの 2 つのインスタンスとして扱われ、異なる相関名 HUSBAND と WIFE を使います。

```
SELECT *
FROM Person HUSBAND, Person WIFE;
```

WITH テーブル・ヒント　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・テーブルとテンポラリ・テーブルでのみ使用できます。

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。

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、無視するデータベース・サーバを指示します。独立性レベル 1 でのみ使用されます。結果は、オプティマイザが使用する最適化方式によって変わります。たとえば、クエリの場合にヒントがテーブルのサブセットのみに指定されている場合です。
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。

テーブル・ヒント	説明
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。
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。「意図的ロック」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
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。「書き込みロック」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

独立性レベルの詳細については、「独立性レベルと一貫性」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

Mobile Link 同期で READPAST を使用
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 - サーバ管理』
- 「download_delete_cursor テーブル・イベント」『Mobile Link - サーバ管理』
- 「upload_fetch テーブル・イベント」『Mobile Link - サーバ管理』

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。「Mobile Link 独立性レベル」『Mobile Link - サーバ管理』を参照してください。

- **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。「optimization_goal オ

プション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

備考

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。

ビューと抽出テーブル
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抽出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 句を使用できません。

パーミッション

なし。

関連する動作

なし。

参照

- 「DELETE 文」 497 ページ
- 「SELECT 文」 669 ページ
- 「UPDATE 文」 728 ページ
- 「ジョイン：複数テーブルからのデータ検索」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** コア機能。ただし、例外があります。KEY JOIN はベンダ拡張です。FULL OUTER JOIN と NATURAL JOIN はコア SQL に含まれていない SQL／基本機能です。READPAST テーブル・ヒントはベンダ拡張です。LATERAL (*table-expression*) はベンダ拡張です (ただし、LATERAL (*query-expression*) は ANSI SQL 規格に機能 T491 として記載されています)。抽出テーブルは機能 F591 です。FROM 句のプロシージャ (テーブル関数) は機能 T326 です。共通テーブル式は機能 T121 です。再帰テーブル式は機能 T131 です。FROM 句は複雑なため、個々の句を標準に照らしてチェックしてください。

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
...
```

次のクエリは、クエリ内での抽出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;

SELECT *
FROM Customers WITH( readpast )
WHERE State = 'NY';
```

GET DATA 文 [ESQL]

この文は、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・データを取得するために使用します。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。「SET 文」 677 ページを参照してください。

構文

GET DATA *cursor-name*
COLUMN *column-num*
OFFSET *start-offset*
[**WITH TEXTPTR**]
USING DESCRIPTOR *sqlda-name* | **INTO** *hostvar*, ⋯

cursor-name : *identifier*, or *hostvar*

column-num : *integer* or *hostvar*

start-offset : *integer* or *hostvar*

sqlda-name : *identifier*

パラメータ

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省略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。

ターゲットのホスト変数のインジケータ値は単精度整数です。このため、インジケータ値にはトランケートされたバイト数が入るとはかぎりません。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 または LONG VARCHAR 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。

WITH TEXTPTR 句　WITH TEXTPTR 句がある場合、テキスト・ポインタが 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 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。

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。

パーミッション

なし。

関連する動作

なし。

参照

- 「FETCH 文 [ESQL] [SP]」 543 ページ
- 「READTEXT 文 [T-SQL]」 639 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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;
```


GET DESCRIPTOR 文 [ESQL]

この文は、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。

構文

GET DESCRIPTOR *descriptor-name*
{ *hostvar* = **COUNT** | **VALUE** { *integer* | *hostvar* } *assignment*, ⋯ }

assignment :
hostvar = **TYPE** | **LENGTH** | **PRECISION** | **SCALE** | **DATA**
| **INDICATOR** | **NAME** | **NULLABLE** | **RETURNED_LENGTH**

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 を実行すると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]」 301 ページ
- 「DEALLOCATE DESCRIPTOR 文 [ESQL]」 486 ページ
- 「SET DESCRIPTOR 文 [ESQL]」 684 ページ
- 「SQLDA (SQL descriptor area)」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** コア機能。

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 );
}
```

詳細例については、「ALLOCATE DESCRIPTOR 文 [ESQL]」 301 ページを参照してください。

GET OPTION 文 [ESQL]

この文を使用すると、オプションの現在の設定を取得できます。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*, or *hostvar*

option-name : *identifier*, *string*, or *hostvar*

hostvar : indicator variable allowed

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に対するものです。これはユーザ個人の設定であるか、または、接続しているユーザ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。

関連する動作

なし。

参照

- 「SET OPTION 文」 686 ページ
- 「システム・プロシージャ」 865 ページ
- 「CONNECTION_PROPERTY 関数 [システム]」 123 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

この文は、ラベルの付いた文に制御を分岐するために使用します。

構文

label : **GOTO** *label*

備考

Transact-SQL プロシージャ、トリガ、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。

パーミッション

なし。

関連する動作

なし。

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

例

次の Transact-SQL バッチは、[サーバ・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 文

この文は、新しいユーザ ID の作成、指定したユーザへのパーミッションの付与または変更、パスワードの作成または変更に使用します。

構文 1

GRANT CONNECT TO *userid*, ⋯
[**AT** *starting-id*]
[**IDENTIFIED BY** *password*, ⋯]

構文 2

GRANT *permission*, ⋯
TO *userid*, ⋯

permission :
DBA
| **BACKUP**
| **VALIDATE**
| **GROUP**
| **MEMBERSHIP IN GROUP** *userid*, ⋯
| [**RESOURCE** | **ALL**]

構文 3

GRANT *permission*, ⋯
ON [*owner.*]*table-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

GRANT EXECUTE ON [*owner.*]*procedure-name*
TO *userid*, ⋯

構文 5

GRANT INTEGRATED LOGIN TO *user-profile-name*, ⋯
AS USER *userid*

構文 6

GRANT KERBEROS LOGIN TO *client-Kerberos-principal*, ⋯
AS USER *userid*

パラメータ

CONNECT TO　新しいユーザを作成します。GRANT CONNECT を使うと、どのユーザ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、他のユーザはグループ・ユーザ 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 ID には、有効な ID を使用します。詳細については、「識別子」 7 ページを参照してください。ユーザ ID とパスワードに使用できない文字列は次のとおりです。

- 空白スペース、一重引用符、または二重引用符で始まる名前
- 空白スペースで終わる名前
- セミコロンを含む名前

パスワードは有効な ID (「識別子」 7 ページを参照してください)、または一重引用符で囲んだ文字列 (最高で 255 バイト) にすることができます。パスワードでは大文字と小文字が区別されます。パスワードには 7 ビット ASCII 文字を使用することをおすすめします。それ以外の文字を使用すると、データベース・サーバがクライアントの文字セットを UTF-8 に変換できない場合、パスワードが機能しないことがあります。

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 ID とパスワードを指定することはできません。「verify_password_function オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

AT starting-id　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 ID に使用する内部数値を指定します。

この句は、アンロード・ユーティリティでの使用を主要な目的として実装されています。

DBA　データベース管理者権限はユーザにパーミッションを与え、すべてのことができるようにします。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。

BACKUP　BACKUP 権限は、ユーザにデータベースをバックアップするパーミッションを与えます。

VALIDATE　VALIDATE 権限は、多様な VALIDATE 文がサポートする検証操作を実行するパーミッションをユーザに与えます。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、チェックサムの検証などです。結果として、ユーザは 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。

GROUP　ユーザがメンバを持てるようにします。「グループの管理」 『SQL Anywhere サーバ - データベース管理』を参照してください。

MEMBERSHIP IN GROUP　ユーザがグループからテーブル・パーミッションを継承し、テーブル名を指定しなくても、グループが作成したテーブルを参照できるようにします。「グループの管理」『SQL Anywhere サーバ - データベース管理』を参照してください。

GRANT 文の構文 3 を使って、個々のテーブルまたはビューに対するパーミッションを付与します。テーブル・パーミッションは個々に指定することも、ALL を使用して一度に 6 つのすべてのパーミッションを付与することもできます。

RESOURCE　ユーザがテーブルやビューを作成できるようにします。構文 2 では、ALL は Sybase Adaptive Server Enterprise に互換性のある RESOURCE と同じです。

ALL　構文 3 では、 ALTER、DELETE、INSERT、REFERENCES、SELECT、UPDATE の各パーミッションを付与します。

ALTER　ユーザは、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　ユーザは、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　ユーザは、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 [(column-name, …)]　ユーザは、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、ユーザはそれらのカラムだけを参照できます。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 [(column-name,…)]　ユーザは、指定したビューまたはテーブルの情報を見ることができます。カラム名を指定する場合、ユーザはそれらのカラムだけを参照できます。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 [(column-name,…)]　ユーザは、指定したビューまたはテーブルのローを更新できます。カラム名を指定する場合、ユーザはそれらのカラムだけを更新できます。

FROM　FROM *userid* を指定すると、その *userid* は付与者のユーザ 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。

備考

GRANT 文を使って、データベース・パーミッションを個々のユーザ ID とグループに付与します。ユーザとグループを作成ときにも使用します。

WITH GRANT OPTION を指定する場合、指定した名前のユーザ ID にもパーミッションが与えられ、他のユーザ ID に同じパーミッションを GRANT (付与) できます。グループのメンバは、グループに付与されている WITH GRANT OPTION を継承しません。

GRANT 文の構文 4 を使って、プロシージャ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 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・コンピュータにログインできたユーザは、ユーザ 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、*ドメイン¥ユーザ名*の形式にすることができます。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 ID にはパスワードを指定する必要があります。「統合化ログインの使用方法」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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 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 (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) は、ユーザ 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 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、*ユーザインスタンス@REALM* にします。この*インスタンス* 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 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 ID (統合化ログイン用と同じ Guest データベースのユーザ ID) を使用して接続します。

Kerberos 認証の詳細については、「Kerberos 認証の使用」 『SQL Anywhere サーバ - データベース管理』を参照してください。

パーミッション

構文 1 または 2　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。別のユーザのパスワードを変更する場合 (DBA 権限を持って) は、データベースに他のユーザが接続していないことが必要です。

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　プロシージャの所有者であるか、DBA 権限が必要です。

構文 5 または 6　DBA 権限が必要です。

関連する動作

オートコミット。

参照

- 「REVOKE 文」 655 ページ

標準と互換性

- **SQL/2003**　構文 3 はコア機能です。構文 4 は永続的ストアド・モジュール機能です。他の構文はベンダ拡張です。

例

データベースに 2 つの新しいユーザを作ります。

```
GRANT
CONNECT TO Laurel, Hardy
IDENTIFIED BY Stan, Ollie;
```

Laurel というユーザに Employee 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 Hardy に Calculate_Report プロシージャの実行を許可します。

```
GRANT EXECUTE ON Calculate_Report
TO Hardy;
```

GRANT CONSOLIDATE 文 [SQL Remote]

この文は、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の識別に使用します。

構文

GRANT CONSOLIDATE
TO *userid*
TYPE *message-system*, …
ADDRESS *address-string*, …
[**SEND** { **EVERY** | **AT** } *hh*:*mm*:*ss*]

message-system:
FILE | **FTP** | **MAPI** | **SMTP** | **VIM**

address: *string*

パラメータ

userid　パーミッションが付与されるユーザのユーザ 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。各データベースが保持できる CONSOLIDATE パーミッションを持つユーザ ID は 1 つだけです。1 つのデータベースを複数の統合データベースに対して共通のリモート・データベースとして指定できません。

統合ユーザはメッセージ・システムにより識別され、統合ユーザ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ディレクトリです。

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 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 文が発行された状態でリモート・データベースが作成されます。

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。

注意
VIM および MAPI のサポートは廃止されました。

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

- 「GRANT PUBLISH 文 [SQL Remote]」 572 ページ
- 「GRANT REMOTE 文 [SQL Remote]」 573 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」 658 ページ

例

```
GRANT CONSOLIDATE TO con_db
TYPE mapi
ADDRESS 'Consolidated Database';
```


GRANT PUBLISH 文 [SQL Remote]

この文は、現在のデータベースのパブリッシャの識別に使用します。

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 ID により識別され、「パブリッシャ」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・ユーザ ID を識別します。

PUBLISH 権限は 1 つのユーザ ID にしか割り当てられません。PUBLISH 権限を持つユーザ ID は、特殊定数 CURRENT PUBLISHER で識別します。次のクエリは、現在のパブリッシャを識別します。

```
SELECT CURRENT PUBLISHER;
```

パブリッシャがない場合、この特殊定数の値は NULL になります。

現在のパブリッシャの特殊定数を、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を削除し、それから GRANT PUBLISH 文を使って新しいパブリッシャを作成します。

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

- 「GRANT PUBLISH 文 [SQL Remote]」 572 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」 570 ページ
- 「REVOKE PUBLISH 文 [SQL Remote]」 659 ページ
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」 453 ページ

例

```
GRANT PUBLISH TO publisher_ID;
```

GRANT REMOTE 文 [SQL Remote]

この文は、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の識別に使用します。これらはリモート・ユーザといいます。

構文

GRANT REMOTE TO *userid*, …
TYPE *message-system*, …
ADDRESS *address-string*, …
[**SEND** { **EVERY** | **AT** } *send-time*]

パラメータ

userid　パーミッションを付与されるユーザのユーザ 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
- MAPI
- SMTP
- VIM

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* のフォーム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・ユーザはメッセージ・システムにより識別され、統合ユーザ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ディレクトリです。

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 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・ユーザに毎日 (SEND AT)、リモート・サーバに 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。

注意
VIM および MAPI のサポートは廃止されました。

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

- 「GRANT PUBLISH 文 [SQL Remote]」 572 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」 661 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」 570 ページ
- 「REMOTE パーミッションと CONSOLIDATE パーミッションの付与と取り消し」 『SQL Remote』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

- 次の文は、REMOTE パーミッションをユーザ SamS に付与し、MAPI E-mail システムを使用し、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 mapi
ADDRESS 'Singer, Samuel'
SEND EVERY '02:00';
```

GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]

この文を使用して、ユーザ ID に REMOTE DBA 権限を付与します。

構文

GRANT REMOTE DBA
TO *userid*, ⋯
IDENTIFIED BY *password*

備考

この文は、同期ユーザに適した DBA 権限の制限されたセットを付与します。REMOTE DBA 権限には、完全な DBA 権限を付与しないようにします。これにより、DBA ユーザ ID やパスワードの分配に関連するセキュリティ問題を回避します。

Mobile Link では、SQL Anywhere 同期クライアント (dbmlsync) に対して REMOTE DBA 権限を付与することをおすすめします。「dbmlsync のパーミッション」 『Mobile Link - クライアント管理』を参照してください。

SQL Remote で REMOTE DBA 権限を使用すると、Message Agent がフル・アクセス権を使用してデータベースにアクセスし、メッセージに変更を加えることができるようになります。

SQL Remote の場合、REMOTE DBA 権限には次のプロパティが設定されます。

- Message Agent からの接続でない場合は、パーミッションを付与しません。REMOTE DBA 権限を付与されたユーザ 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 ユーザのユーザ 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 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。
- Message Agent から接続した場合は、フル DBA パーミッションを付与します。

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

- Mobile Link：「同期の開始」 『Mobile Link - クライアント管理』
- SQL Remote：「Message Agent とレプリケーション・セキュリティ」 『SQL Remote』
- 「REVOKE REMOTE DBA 文 [SQL Remote]」 662 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

GROUP BY 句

この句を使用して、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

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 が基礎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。「GROUPING 関数 [集合]」 173 ページを参照してください。

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。

GROUPING セットで空のかっこを使用する場合の例など詳細については、「空のグループ化指定の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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。

ROLLUP 演算の詳細については、「ROLLUP の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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。

ROLLUP 演算の詳細については、「CUBE の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

WITH ROLLUP 句　これは ROLLUP 句の代わりになる構文です。また、T-SQL との互換性のために用意されています。

WITH CUBE く　これは CUBE 句の代わりになる構文です。また、T-SQL との互換性のために用意されています。

備考

GROUP BY 句を使用している場合、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・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。

参照

- 「SELECT 文」 669 ページ
- 「GROUP BY 句の拡張」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　GROUP BY 句はコア機能ですが、GROUPING SETS、ROLLUP、CUBE はコア SQL 機能ではありません。たとえば、ROLLUP 句は機能 T431 の一部です。WITH ROLLUP と WITH CUBE はベンダ拡張です。

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;
```

HELP 文 [Interactive SQL]

この文は、Interactive SQL 環境でヘルプを表示するために使用します。

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。

標準と互換性

- **SQL/2003**　ベンダ拡張。

IF 文

この文は、SQL 文の条件付き実行を制御するために使用します。

構文

IF *search-condition* **THEN** *statement-list*
[**ELSEIF** { *search-condition* | *operation-type* } **THEN** *statement-list*] …
[**ELSE** *statement-list*]
END IF

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。

IF 文は IF 式とは異なります。
IF 文の構文と IF 式の構文を混同しないでください。
IF 式の詳細については、「IF 式」 17 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「BEGIN 文」 356 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「探索条件」 20 ページ

標準と互換性

- **SQL/2003** 永続的ストアド・モジュール機能。

例

次のプロシージャは、IF 文の使い方を示します。

```
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
```

IF 文 [T-SQL]

この文は、Watcom-SQL IF 文の代わりに、条件による SQL 文の実行を制御するために使用します。

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 には THEN 文がありません。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/2003** 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]

この文は、SQL プリプロセッサによってスキャンされたソース・プログラムにファイルをインクルードするために使用します。

構文

INCLUDE *file-name*

file-name : **SQLDA** | **SQLCA** | *string*

備考

INCLUDE 文は、C プリプロセッサ #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 が特に認識されます。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 プリプロセッサが宣言する変数 (プログラマには見えません) 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/2003** コア機能。

INPUT 文 [Interactive SQL]

この文は、外部ファイルまたはキーボードからデータベースにデータをインポートするために使用します。

構文

INPUT INTO [*owner*.]*table-name*
[**FROM** *file-name* | **PROMPT**]
[**FORMAT** *input-format*]
[**ESCAPE CHARACTER** *character*]
[**ESCAPES** { **ON** | **OFF** }
[**BY ORDER** | **BY NAME**]
[**DELIMITED BY** *string*]
[**COLUMN WIDTHS (** *integer*, ⋯**)**]
[**NOSTRIP**]
[**(** *column-name*, ⋯ **)**]
[**ENCODING** *encoding*]

input-format :
ASCII | **DBASE** | **DBASEII** | **DBASEIII**
| **EXCEL** | **FIXED** | **FOXPRO** | **LOTUS**

encoding : *identifier* or *string*

パラメータ

INTO 句 データを入力するテーブルの名前です。

FROM 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。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。したがって、ファイル c:¥temp¥input.dat から Employee 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';
```

- パス名は、Interactive SQL が実行されているコンピュータからの相対です。

PROMPT 句 PROMPT 句では、ユーザ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、ユーザが新規ローの値を入力できるダイアログが表示されます。コマンド・ラインで Interactive SQL を実行している場合、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。

FORMAT 句 値の各セットは、FORMAT 句で指定したフォーマットの中、または FORMAT 句を指定しない場合は SET OPTION *input_format* 文で設定したフォーマットの中に入れてください。

特定のファイル・フォーマットには、カラム名と型に関する情報が含まれています。データベース・テーブルが存在していない場合、INPUT 文はこの情報を使ってデータベース・テーブルを作成します。これは、データベースにデータをロードする非常に簡単な方法です。テーブルの作成に十分な情報を持つフォーマットは、DBASEII、DBASEIII、EXCEL、FOXPRO、LOTUS です。

コマンド・ファイルからの入力は、END がある行で終了します。ファイルからの入力はファイルの最後で終了します。

使用できる入力フォーマットは、次のとおりです。

- **ASCII** 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で区切られているものとみなされます。アルファベットの文字列をアポストロフィ (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。

  ファイルに値が NULL の可能性があることを示すエントリがある場合、そのデータは 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、文字列カラムには空の文字列が挿入されます。

- **DBASE** このファイルは、dBASE II または dBASE III フォーマットです。Interactive SQL はどちらのフォーマットであるかを、ファイルの中の情報に基づいて決定しようとします。このテーブルが存在しない場合は作成されます。

- **DBASEII** このファイルは dBASE II フォーマットです。このテーブルが存在しない場合は作成されます。

- **DBASEIII** このファイルは dBASE III フォーマットです。このテーブルが存在しない場合は作成されます。

- **EXCEL** 入力ファイルは Microsoft Excel 2.1 のフォーマットです。このテーブルが存在しない場合は作成されます。

- **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・カラムの型の値に必要な文字の最大値と同じにしてください。

  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。

- **FOXPRO** このファイルは FoxPro フォーマットです。このテーブルが存在しない場合は作成されます。

- **LOTUS** このファイルは Lotus WKS フォーマットのワークシートです。INPUT は、Lotus WKS フォーマット・ワークシートの第 1 行はカラム名であるとみなします。テーブルがない場合は、テーブルが作成されます。新規テーブルの定義に使用されるデータ型は、Lotus ワークシートのセル値に基づいて選択されます。

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。

改行文字は ¥n との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 '!'
```

ESCAPES 句　ESCAPES をオン (デフォルト) にすると、データベース・サーバ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 をオフにすると、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。

BY 句　BY 句は、入力ファイルのカラムをテーブル・カラムと一致させる基準として、ユーザがリストの並び順 (デフォルトの ORDER) または名前 (NAME) を指定できるようにします。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 は、それを持つフォーマットにだけ許可されています。これらは、テーブルの自動作成を許可する DBASEII、DBASEIII、EXCEL、FOXPRO、LOTUS と同じフォーマットです。

DELIMITED BY 句　DELIMITED BY 句を使用すると、ASCII 入力フォーマットの中でデリミタ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はカンマです。

COLUMN WIDTHS 句　COLUMN WIDTHS を指定できるのは、FIXED フォーマットの場合だけです。これは入力ファイルのカラムの幅を指定します。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。FIXED フォーマットの LONG VARCHAR または BINARY データを挿入する場合は、この句を使用しないでください。

NOSTRIP 句　通常、ASCII 入力フォーマットの場合、後続ブランクを引用符の付いていない文字列から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できません。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。

ENCODING 句　*encoding* 引数では、ファイルの読み込みに使用されるコードを指定できます。ENCODING 句は、ASCII フォーマットでのみ使用できます。

SQL Anywhere でサポートされるエンコードのリストを取得する方法については、「サポートされている文字セット」『SQL Anywhere サーバ - データベース管理』を参照してください。

Interactive SQL の場合、*encoding* が指定されていないと、次のようにリストで先に出現するエンコード値を後で出現するエンコード値より優先することで、ファイルの読み込みに使用するエンコードを判断します。

- 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)
- Interactive SQL の開始時に -codepage で指定されたエンコード
- Interactive SQL が動作しているコンピュータのデフォルトのエンコード

Interactive SQL とエンコードの詳細については、「default_isql_encoding オプション [Interactive SQL]」『SQL Anywhere サーバ - データベース管理』を参照してください。

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、[対話型入力] ウィンドウを通してユーザから (PROMPT を指定する場合)、またはファイルから (FROM *file-name* を指定する場合) 読み込まれます。どちらも指定しない場合は、INPUT 文を含むコマンド・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。

[SQL 文] ウィンドウ枠から入力を直接読み取る場合、INPUT 文の末尾にレコードを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 文は、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。

いずれかの入力フォーマットに対してカラム・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。デフォルトでは、INPUT 文は、入力ファイルのカラムの値がデータベース・テーブル定義の中と同じ順序で表示されるものとみなします。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 ASCII 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 ASCII
(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することができます (「SET OPTION 文」を参照してください)。文字列値が INPUT 上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 は 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 コマンドなので、複合文 (IF など) やストアド・プロシージャには使用できません。

「プロシージャ、トリガ、イベント、バッチで使用できる文」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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

- 「OUTPUT 文 [Interactive SQL]」 623 ページ
- 「INSERT 文」 590 ページ
- 「UPDATE 文」 728 ページ
- 「DELETE 文」 497 ページ
- 「SET OPTION 文」 686 ページ
- 「LOAD TABLE 文」 603 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次は、ASCII テキスト・ファイルからの INPUT 文の例です。

```
INPUT INTO Employees
FROM new_emp.inp
FORMAT ASCII;
```

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** }] }]
VALUES (*expression* | **DEFAULT**, ⋯ **)**
[**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**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

パラメータ

WITH AUTO NAME 句　WITH AUTO NAME は、構文 2 だけに適用されます。WITH AUTO NAME を指定すると、SELECT 文の項目名によってデータの所属カラムが決まります。SELECT 文の項目には、カラム参照かエイリアスの式を指定してください。SELECT 文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。

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。

ON EXISTING 句を指定すると、データベース・サーバ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。

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 パラメータと DEFAULTS OFF パラメータは、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、AUTOINCREMENT 順にスキップされた値が設定されます。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 ユーザに設定されます。

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 '*option-value*'** MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザがクエリを処理するときに実体化ビュー (Materialized View) 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。

「materialized_view_optimization オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **FORCE OPTIMIZATION** 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省略されます。ただし、コストベースの最適化を実行したい場合もあります。たとえば、クエリ処理時に実体化ビュー (Materialized View) 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、ビューのマッチング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、プロシージャ内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャに対するすべての呼び出しにオプティマイザ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。

  単純なクエリとビューのマッチングに関する詳細については、「クエリ処理のフェーズ」『SQL Anywhere サーバ - SQL の使用法』と「実体化ビュー (Materialized View) によるパフォーマンスの向上」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- ***option-name = option-value*** 該当する文に対してのみ、パブリック・オプションやテンポラリ・オプションの設定よりも優先されるオプション設定を指定します。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 [互換性]」『SQL Anywhere サーバ - データベース管理』
  - 「max_query_tasks オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

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。

構文 1 指定された式のカラム値を使用して、1 つのローを挿入します。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、通常、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一致します。

ビューへの挿入ができるのは、ビューを定義するクエリ指定が更新可能で、FROM 句にテーブルが 1 つしか指定されていない場合です。

次の要素を含むクエリ式またはクエリ指定で構成されるビューは、本来は更新不可能です。

- DISTINCT 句
- GROUP BY 句
- 集合関数
- ベース・テーブル以外の *select-list* 項目

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。したがって、テーブルの中へ挿入される文字列 Value は、常に大文字の V と残りの英字が小文字で格納されます。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 などと同じとみなされます。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。

パフォーマンスを改善するためのヒント
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。「ALTER TABLE 文」 336 ページを参照してください。

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

- 「INPUT 文 [Interactive SQL]」 585 ページ
- 「LOAD TABLE 文」 603 ページ
- 「UPDATE 文」 728 ページ
- 「DELETE 文」 497 ページ
- 「PUT 文 [ESQL]」 633 ページ

標準と互換性

- **SQL/2003**　コア機能。INSERT … ON EXISTING はベンダ拡張です。

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、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) );
INSERT
INTO DepartmentHead (ManagerName, DepartmentName)
SELECT GivenName || ' ' || Surname, DepartmentName
FROM Employees JOIN Departments
ON EmployeeID = DepartmentHeadID;
```

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) );
INSERT
INTO DepartmentHead WITH AUTO NAME
SELECT GivenName || ' ' || Surname AS ManagerName,
    DepartmentName
FROM Employees JOIN Departments
ON EmployeeID = DepartmentHeadID;
```

テーブル MyTable を作成し、WITH AUTO NAME 構文を使用してデータを挿入します。

```
CREATE TABLE MyTable(
    pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
    TableName CHAR(128),
    TableNameLen INT );
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;
```

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 );
```

INSTALL JAVA 文

この文は、データベース内で Java クラスを使用できるようにします。

構文

INSTALL JAVA
[NEW | UPDATE]
[JAR *jar-name* **]**
FROM { FILE *file-name* **|** *expression* **}**

パラメータ

NEW | UPDATE キーワード　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、*file-name* 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。

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。

file-name 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。

file-name には、クラス・ファイルまたは jar ファイルのどちらを特定してください。

FROM 式句　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、すべてのユーザがすべての方法で参照できます。

Windows CE ではサポートされません。

参照

- 「REMOVE JAVA 文」 646 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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。

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-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

パラメータ

注意
query-block で FOR、FOR XML、WITH または OPTION の句を使用することはできません。

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 '*option-value*'** MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザがクエリを処理するときに実体化ビュー (Materialized View) 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。「materialized_view_optimization オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **FORCE OPTIMIZATION** 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省略されます。ただし、コストベースの最適化を実行したい場合もあります。たとえば、クエリ処理時に実体化ビュー (Materialized View) 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、ビューのマッチング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、プロシージャ内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャに対するすべての呼び出しにオプティマイザ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。

単純なクエリとビューのマッチングに関する詳細については、「クエリ処理のフェーズ」『SQL Anywhere サーバ - SQL の使用法』と「実体化ビュー (Materialized View) によるパフォーマンスの向上」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- ***option-name = option-value*** 該当する文に対してのみ、パブリック・オプションやテンポラリ・オプションの設定よりも優先されるオプション設定を指定します。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 [互換性]」『SQL Anywhere サーバ - データベース管理』
  - 「max_query_tasks オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

備考

複数のクエリ・ブロックの結果セット間の共通部分は、INTERSECT または INTERSECT ALL を使用して 1 つの結果として取得できます。INTERSECT DISTINCT は INTERSECT と同じです。

クエリ・ブロックでは、それぞれ select リストの中の項目数が同じになるようにしてください。

INTERSECT の結果は INTERSECT ALL と同じです。ただし、INTERSECT を使用するときに、重複するローが削除されてから結果セット間の共通部分が計算される場合は除きます。

2 つの select リスト中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。UNION の最初の *query-block* は、ORDER BY 句と一致する名前を判断するために使用します。

表示されるカラム名は、最初の *query-block* に対して表示されるカラム名と同じです。結果セットのカラム名をカスタマイズするには、*query-block* で WITH 句を使用するという方法もあります。

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」 529 ページ
- 「UNION 文」 720 ページ

標準と互換性

- **SQL/2003** F302 の機能。

例

INTERSECT の使用例については、「集合演算子と NULL」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

LEAVE 文

この文は、複合文またはループから出るために使用します。

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。

プロシージャまたはトリガの本文である複合文は、プロシージャまたはトリガ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」 614 ページ
- 「FOR 文」 547 ページ
- 「BEGIN 文」 356 ページ
- 「プロシージャ、トリガ、バッチの使用」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** 永続的ストアド・モジュール機能。

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

内部でのみ使用されます。この文は、統計情報をシステム・テーブル ISYSCOLSTAT 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

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 権限が必要です。

関連する動作

なし。

参照

- 「ISYSCOLSTAT システム・テーブル」 753 ページ
- 「アンロード・ユーティリティ (dbunload)」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

LOAD TABLE 文

この文は、外部ファイルからデータベース・テーブルの中へバルク・データをインポートするために使用します。*挿入はログ・ファイルに記録されません。*そのため、エラーによってデータが失われる危険があり、この文は SQL Remote や Mobile Link リモート・データベースでは使用できません。

構文

LOAD [**INTO**] **TABLE** [*owner.*]*table-name* [**(** *column-name*, ⋯ **)**]
FROM *file-name*
[*load-option* ⋯]
[*statistics-limitation-options*]

load-option :
CHECK CONSTRAINTS { **ON** | **OFF** }
| **COMMENTS INTRODUCED BY** *comment-prefix*
| **COMPUTES** { **ON** | **OFF** }
| **DEFAULTS** { **ON** | **OFF** }
| **DELIMITED BY** *string*
| **ENCODING** *encoding*
| **ESCAPE CHARACTER** *character*
| **ESCAPES** { **ON** | **OFF** }
| **FORMAT** { **ASCII** | **BCP** }
| **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** }

statistics-limitation-options :
STATISTICS { **ON** [**ALL COLUMNS**]
| **OFF**
| **ON KEY COLUMNS**
| **ON (** *column-list* **)** }

file-name : *string* | *variable*

comment-prefix : *string*

encoding : *string*

パラメータ

カラム名　DEFAULTS オプションが 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 入力不可能なカラムがカラム・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、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 がオンかオフ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。

FROM オプション　*file-name-string* は文字列としてデータベース・サーバに渡されます。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。したがって、ファイル c:¥temp¥input.dat から Employee 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。別のコンピュータのデータベース・サーバ上で文を実行している場合、ディレクトリ名はそのデータベース・サーバ・コンピュータのディレクトリを参照し、クライアント・コンピュータのディレクトリは参照しません。
- UNC パス名を使用すると、データベース・サーバ以外のコンピュータ上のファイルからデータをロードできます。

CHECK CONSTRAINTS オプション　このオプションはデフォルトで ON に設定されますが、アンロード・ユーティリティはこのオプションを 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、検査制約は無効になります。この機能は、データベースを再構築するときなどに役立ちます。テーブルにまだ作成されていないユーザ定義関数を呼び出す検査制約があると、このオプションが OFF に設定されていない場合、再構築は失敗します。

COMMENTS INTRODUCED BY オプション　このオプションを使用すると、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ことができます。使用すると、LOAD TABLE は文字列 *comment-prefix* で始まる行を無視します。

この例では、// 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 '//' ...
```

コメントは新しい行の先頭にのみ指定できます。

COMMENTS INTRODUCED BY オプションを省略する場合、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。

COMPUTES オプション　デフォルトでは、COMPUTES は ON です。COMPUTES を ON に設定すると、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。このオプションは、たとえば、データベースを再構築するとき、テーブルの中に、まだ作成されていないユーザ定義関数を呼び出す計算カラムが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 オプション　デフォルトでは、DEFAULTS は OFF です。DEFAULTS オプションが 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 オプションが OFF で、NULL 入力不可能なカラムがカラム・リストから省かれている場合は、データベース・サーバは、空の文字列をカラムの型に変換しようとします。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。

DELIMITED BY オプション　デフォルトのカラム・デリミタ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。同じフォーマット要件が、他の SQL 文字列に適応されます。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 オプション　ENCODING オプションを使用して、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します。ロードするファイルのデータは、指定された文字エンコードで適切にエンコードされている必要があります。ENCODING が指定されていない場合、データベースの文字エンコードが使用され、変換は実行されません。

SQL Anywhere でサポートされるエンコードのリストを取得する方法については、「サポートされている文字セット」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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。「on_charset_conversion_failure オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

次の例では、UTF-8 文字エンコードを使用してデータをロードします。

```
LOAD TABLE mytable FROM 'mytable_data_in_utf8.dat' ENCODING 'UTF-8';
```

ESCAPE CHARACTER オプション　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 '!'
```

エスケープ文字として使用できるのは、1 つの 1 バイト文字だけです。

ESCAPES オプション　ESCAPES を ON (デフォルト) にすると、データベース・サーバ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。改行文字は ¥n との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。

FORMAT オプション　ASCII を選択すると、入力行は ASCII 文字であり、1 行あたり 1 つのローで構成され、カラム・デリミタ文字列によって値が区切られます。BCP を選択すると、BLOB を含む Adaptive Server Enterprise 生成の BCP アウト・ファイルをインポートできます。

HEXADECIMAL オプション　デフォルトでは、HEXADECIMAL は ON です。HEXADECIMAL が 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。*n* はそれぞれ 16 進数字です。マルチバイト文字セットを扱う場合は、HEXADECIMALON を使用することが重要です。

HEXADECIMAL オプションを使用する場合は、FORMAT ASCII オプションのみ指定してください。

ORDER オプション　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。「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

PCTFREE オプション　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。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 ー 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 の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

QUOTE オプション　QUOTE 句は ASCII データベースのみ用です。*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 オプション　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。引用符文字はアポストロフィ (一重引用符) または二重引用符のいずれかです。文字列の中で最初に出てくるこのような文字は、その文字列の引用符文字として処理されます。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 ON を指定して、カラム・デリミタ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みなされません。したがって、次の行はアドレスにカンマがあるかどうかは関係ありません。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インクルードするには、QUOTES を ON にして、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 オプション　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。同じフォーマット要

件が、他の SQL 文字列に適応されます。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 オプション　ファイルの最初の数行を無視するには SKIP オプションを含めます。*integer* 引数では、スキップする行数を指定します。たとえば、このオプションを使用して、カラム見出しを含む行をスキップできます。ロー・デリミタがデフォルト (改行) でない場合、引用符で囲まれた文字列とともに埋め込まれたロー・デリミタ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 オプション　STRIP オプションは、引用符がない値に、値を挿入する前に削除された先行ブランクまたは後続ブランクがあ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**　先行ブランクまたは後続ブランクを削除しません。
- **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**　推奨されなくなった句。STRIP RTRIM と同等です。

WITH CHECKPOINT オプション　デフォルト設定は OFF です。ON に設定すると、文が正常に完了し、ログした後、チェックポイントが発行されます。

WITH CHECKPOINT ON を指定せず、CHECKPOINT 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、リカバリが正しく完了するためには、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なければなりません。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。

警告
データベース・オプション conversion_error を Off に設定すると、不正なデータをロードしてもエラーがレポートされないことがあります。WITH CHECKPOINT ON を指定しない場合、データベースのリカバリが必要となったときには、conversion_error が On (デフォルト値) であれば、リカバリが失敗することがあります。conversion_error を Off に設定し、WITH CHECKPOINT ON を指定せずに、テーブルをロードしないことをおすすめします。

「conversion_error オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

データベースが破壊され、バックアップを使って現在のログ・ファイルを適用する必要がある場合は、このオプションの設定にかかわらずデータ・ファイルが必要となります。

statistics-limitation-options　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、このオプション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

警告
LOAD TABLE は、大量データを高速にロードすることのみを目的とするものです。LOAD TABLE では、トランザクション・ログに個々のローを書き込みません。

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・テーブルとグローバル・テンポラリ・テーブルの場合、コミットが実行されます。ローカル・テンポラリ・テーブルの場合、コミットは実行されません。

UTF-16 または UTF-8 データ・ファイルからデータを読み込むとき、LOAD TABLE はバイト順マーク (BOM) を無視します (マークがある場合でも)。データベース・サーバは、データのバイト順は、データベース・サーバが実行されているコンピュータと同じであることを前提とします。

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、グローバル・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、LOAD TABLE を使用できます。

FORMAT ASCII 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、1,,'Fred', という値を含むファイルに 3 つの値が予想される場合、挿入される値は 1、NULL、'Fred' です。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は、NULL 値と扱われます。たとえば、1, ,'Fred', を含むファイルに 3 つの値が予想される場合、値 1、NULL、'Fred' が挿入されます。他の値はすべて NULL 以外と扱われます。たとえば、'' (一重引用符が 2 つ) は空の文字列です。'NULL' は 4 文字の文字列です。

LOAD TABLE でロードしていないカラムが 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 でロードされたカラムが 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。

LOAD TABLE 文にカラム・リストが含まれる場合、カラム・リストに指定されていないカラムは次のようにロードされます。

- DEFAULT ON が指定されると、カラム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。
- カラムに DEFAULT 値がなく、NULL を許容している場合、NULL が使用されます。
- カラムに DEFAULT 値がなく、NULL を許容していない場合、カラムのデータ型に応じて、ゼロ (0) または空の文字列 ('') が使用されるかエラーが返されます。

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、オプティマイザによって使用されます。オプティマイザがカラム統計を使用する方法の詳細については、「オプティマイザの推定とカラム統計」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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。「DROP STATISTICS 文」 523 ページを参照してください。
- カラムに対して LOAD TABLE を実行するときにカラム統計が存在する場合、カラムの統計は再計算されないことに注意してください。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。「カラム統計の更新」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- 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。

「-gl サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』を参照してください。

テーブルに排他ロックを実行してください。

関連する動作

ローカル・テンポラリ・テーブルを除いた自動コミット。

挿入はログ・ファイルに記録されません。そのため、エラーが発生した場合、挿入されたローを回復できないことがあります。また、LOAD TABLE 文は、SQL Remote レプリケーションに関連するデータベースまたは Mobile Link クライアントとして使用されるデータベースでは使用しないでください。これらのテクノロジは、ログ・ファイルを解析して変更をレプリケートするからです。

LOAD TABLE 文は、テーブルに関連したトリガは起動しません。

チェックポイントはオペレーションの開始時に実行されます。オペレーションの終了時に行うチェックポイントは任意です。

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。

参照

- 「UNLOAD TABLE 文」 725 ページ
- 「INSERT 文」 590 ページ
- 「INPUT 文 [Interactive SQL]」 585 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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 ASCII DEFAULTS ON;
```

コマンド SELECT * FROM t は、次の結果セットを返します。

```
this_is_for_column_a, 1, this_is_for_column_c
```


動的に構成されたファイル名を指定し、EXECUTE IMMEDIATE 文を使用して、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 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 TABLE 文

この文は、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をアクセスしたり、修正することを防止するために使用します。

構文

LOCK TABLE *table-name*
[WITH HOLD]
IN { SHARE | EXCLUSIVE } MODE

パラメータ

table-name　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　この句を指定すると、ロックは接続が終了するまで保持されます。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。

SHARE モード　他のトランザクションに対し、テーブルの修正を禁止します。ただし、読み込みアクセスは許可します。SHARE モードで、他のトランザクションが修正中のローに何もロックをかけていない場合、トランザクションはデータを変更できます。

EXCLUSIVE モード　他のトランザクションに対し、テーブルへのアクセスを禁止します。他のトランザクションは、クエリやあらゆる種類の更新を含め、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 t IN EXCLUSIVE MODE などの文を使用してテーブル t を排他的にロックすると、サーバのデフォルト動作では、*t* 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。

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 モードの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

- 「SELECT 文」 669 ページ

- 「sa_locks システム・プロシージャ」 915 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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 Customer テーブルの修正を禁止します。

```
LOCK TABLE Customers
IN SHARE MODE;
```


LOOP 文

この文は、文リストの実行を繰り返すために使用します。

構文

[*statement-label* :]
[**WHILE** *search-condition*] **LOOP**
  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」 547 ページ
- 「CONTINUE 文 [T-SQL]」 378 ページ

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

例

プロシージャ中の 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;
...
```

プロシージャ中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
```


MESSAGE 文

この文は、メッセージを表示するために使用します。

構文

MESSAGE *expression*, …
[**TYPE** { **INFO** | **ACTION** | **WARNING** | **STATUS** }]
[**TO** { **CONSOLE**
  | **CLIENT** [**FOR** { **CONNECTION** *conn_id* | **ALL** }]
  | [**EVENT** | **SYSTEM**] **LOG** }
  [**DEBUG ONLY**]
]

conn_id : integer

パラメータ

TYPE 句 この句は、メッセージ・タイプを指定します。指定できる値は INFO、ACTION、WARNING、および STATUS です。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、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] ボタンのあるメッセージ・ボックス
- **WARNING** [OK] ボタンのあるメッセージ・ボックス
- **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** メッセージを [サーバ・メッセージ] ウィンドウに送信します。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サーバ・ログ・ファイルに送信します。EVENT または SYSTEM を指定すると、メッセージはコンソール、Windows イベント・ログ (イベント・ソース SQLANY 10.0 Admin の下)、UNIX SysLog (SQLANY 10.0 Admin (servername) という名前の下) にも書き込まれます。サーバ・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　メッセージ TO CLIENT では、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。

- **CONNECTION conn_id**　メッセージの受信者の接続 ID を指定します。
- **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。

MESSAGE … TO CLIENT 文を発行するプロシージャは、接続に対応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に、メッセージ・ボックスが表示されません。

```
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
  MESSAGE 'Idle engine' type warning to client;
END;
```

ただし、次の例ではメッセージがサーバ・コンソールに書き込まれます。

```
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
  MESSAGE 'Idle engine' 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、第 2 の接続を使用できます。この接続では、1 つ以上の WAITFOR DELAY 文を実行できます。これらの文は、(ポーリング・アプローチと異なり) サーバまたはネットワーク上のリソースを大量には消費しませんが、アプリケーションはメッセージが送信された直後にメッセージの通知を受信できます。

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が ASA_REGISTER_MESSAGE_CALLBACK 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

- 「CREATE PROCEDURE 文」 423 ページ
- 「debug_messages オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「db_register_a_callback 関数」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

1. 次のプロシージャは、[サーバ・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、データベースの [サーバ・メッセージ] ウィンドウに、文字列「The current date and time (現在の日時)」の次に現在の日時を表示します。

   ```
   CALL message_text();
   ```

2. ODBC でコールバックを登録するには、メッセージ・ハンドラを宣言します。

   ```
   void SQL_CALLBACK my_msgproc(
     VOID * sqlca,
     UNSIGNED CHAR msg_type,
     LONG code,
     UNSIGNED SHORT len,
     CHAR* msg )
   { … }
   ```

   msg が null で終了されていないことに注意してください。この問題を処理するようにアプリケーションを設計する必要があります。

3. SQLSetConnectAttr 関数を呼び出して、宣言されたメッセージ・ハンドラをインストールします。

```
rc = SQLSetConnectAttr(
  dbc,
  SA_REGISTER_MESSAGE_CALLBACK,
  (SQLPOINTER) &my_msgproc, SQL_IS_POINTER );
```

1. Embedded SQL でコールバックを登録するには、最初にメッセージ・ハンドラを宣言します。

```
void SQL_CALLBACK my_msgproc(
  SQLCA * sqlca,
  UNSIGNED CHAR msg_type,
  LONG code,
  UNSIGNED SHORT len,
  CHAR* msg ) // msg is NOT null terminated
{ ... }
```

2. db_register_a_callback 関数を呼び出して、宣言されたメッセージ・ハンドラをインストールします。

```
db_register_a_callback( &sqlca, DB_CALLBACK_MESSAGE, (SQL_CALLBACK_PARM)
&my_msgproc );
```


OPEN 文 [ESQL] [SP]

この文は、事前に宣言したカーソルを開き、データベースからの情報にアクセスするために使用します。

構文

OPEN *cursor-name*
[**USING** [**DESCRIPTOR** *sqlda-name* | *hostvar*, ⋯]]
[**WITH HOLD**]
[**ISOLATION LEVEL** *n*]
[**BLOCK** *n*]

cursor-name : *identifier* or *hostvar*

sqlda-name : *identifier*

パラメータ

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 (「row_counts オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください)、これはおすすめしません。

cursor-name が識別子または文字列によって指定される場合、C プログラムでは OPEN の前に対応する DECLARE CURSOR 文を置きます。ホスト変数によって *cursor-name* を指定する場合、DECLARE CURSOR 文を OPEN 文の前で実行します。

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・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。

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。

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。「ロックの仕組み」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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(「ESQL でのカーソルの使用」『SQL Anywhere サーバ - プログラミング』または「プロシージャとトリガでのカーソルの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください)。

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。「FETCH 文 [ESQL] [SP]」 543 ページを参照してください。

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を実行します。プロシージャ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。

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 文のプロシージャに EXECUTE パーミッションが必要です。

関連する動作

なし。

参照

- 「DECLARE CURSOR 文 [ESQL] [SP]」 489 ページ
- 「RESUME 文」 652 ページ
- 「PREPARE 文 [ESQL]」 629 ページ
- 「FETCH 文 [ESQL] [SP]」 543 ページ
- 「RESUME 文」 652 ページ
- 「CLOSE 文 [ESQL] [SP]」 368 ページ
- 「FOR 文」 547 ページ

標準と互換性

- **SQL/2003** ESQL の使用はコア機能です。プロシージャの使用は、永続的ストアド・モジュール機能です。

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またはトリガからのものです。

```
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]

この文は、現在のクエリ結果をファイルに出力するために使用します。

構文

OUTPUT TO *file-name*
[**APPEND**]
[**VERBOSE**]
[**FORMAT** *output-format*]
[**ESCAPE CHARACTER** *character*]
[**ESCAPES** { **ON** | **OFF** }]
[**DELIMITED BY** *string*]
[**QUOTE** *string* [**ALL**]]
[**COLUMN WIDTHS (***integer, ⋯***)**]
[**HEXADECIMAL** { **ON** | **OFF** | **ASIS** }]
[**ENCODING** *encoding*]

output-format :
ASCII | **DBASEII** | **DBASEIII** | **EXCEL**
| **FIXED** | **FOXPRO** | **HTML** | **LOTUS**
| **SQL** | **XML**

encoding : *string* or *identifier*

パラメータ

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 ASCII、FIXED、または SQL の場合に、APPEND キーワードが有効です。

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。VERBOSE キーワードは、出力フォーマットが ASCII、FIXED、または SQL の場合に有効です。

FORMAT 句　使用できる出力フォーマットは、次のとおりです。

- **ASCII**　出力は ASCII フォーマット・ファイルであり、1 行あたり 1 つのローで構成されます。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。

  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。これはデフォルト出力フォーマットです。

- **DBASEII**　出力は、カラム定義を含む dBASE II 形式のファイルです。最大 32 カラムを出力できることに注意してください。カラム名は 11 文字にトランケートされ、各カラムのデータの各ローは 255 文字にトランケートされます。

- **DBASEIII** 出力は、カラム定義を含む dBASE III 形式のファイルです。最大 128 カラムを出力できることに注意してください。カラム名は 11 文字にトランケートされ、各カラムのデータの各ローは 255 文字にトランケートされます。
- **EXCEL** この出力は Excel 2.1 のワークシートです。ワークシートの最初のローには、カラム・ラベル (または、ラベルが定義されていない場合はカラム名) があります。2 つ目以降のワークシート・ローには、実際のテーブル・データがあります。
- **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中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。
- **FOXPRO** 出力は、カラム定義を含む FoxPro 形式のファイルです。最大 128 カラムを出力できることに注意してください。カラム名は 11 文字にトランケートされます。カラム名は 11 文字にトランケートされ、各カラムのデータの各ローは 255 文字にトランケートされます。
- **HTML** この出力は HTML (Hyper Text Markup Language) フォーマットです。
- **LOTUS** 出力は、Lotus WKS フォーマットのワークシートです。カラム名をワークシートの最初のローとして入れます。(Lotus 1-2-3 のような) 他のソフトウェアがロードできる Lotus WKS フォーマット・ワークシートの最大サイズに、一定の制限があることに注意してください。Interactive SQL のファイル・サイズには制限はありません。
- **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。INPUT 文は、XML をファイル・フォーマットとして受け入れません。

ESCAPE CHARACTER 句 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 '!'
```

改行文字は ¥n との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。

ESCAPES 句 ESCAPES をオン (デフォルト) にすると、データベース・サーバ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 をオフにすると、ソースに記載されているとおりに文字が書き込まれます。

DELIMITED BY 句　DELIMITED BY 句を使用できるのは、ASCII 出力フォーマットの場合のみです。カラムはデリミタ文字列 (デフォルトはカンマ) で区切られます。

QUOTE 句　QUOTE 句を使用できるのは、ASCII 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文字です。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。

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。

HEXADECIMAL 句　HEXADECIMAL 句では、ASCII フォーマットの場合にのみ、バイナリ・データをアンロードする方法を指定します。この句を ON に設定すると、バイナリ・データは 0xabcd フォーマットでアンロードされます。この句を OFF に設定すると、バイナリ・データはアンロード時にエスケープされます (¥xab¥xcd)。ASIS に設定すると、値はそのまま書き込まれます。つまり、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。

ENCODING 句　*encoding* 引数では、ファイルの書き込みに使用されるコードを指定できます。ENCODING 句は、ASCII フォーマットでのみ使用できます。

SQL Anywhere でサポートされるエンコードのリストを取得する方法については、「サポートされている文字セット」『SQL Anywhere サーバ - データベース管理』を参照してください。

Interactive SQL の場合、*encoding* が指定されていないと、次のようにリストで先に出現するエンコード値を後で出現するエンコード値より優先することで、ファイルの書き込みに使用するエンコードを判断します。

- 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)
- Interactive SQL の開始時に -codepage で指定されたエンコード
- Interactive SQL が動作しているコンピュータのデフォルトのエンコード

Interactive SQL とエンコードの詳細については、「default_isql_encoding オプション [Interactive SQL]」『SQL Anywhere サーバ - データベース管理』を参照してください。

備考

OUTPUT 文は、現在のクエリが取り出した情報をファイルにコピーします。

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。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_format オプション [Interactive SQL]」『SQL Anywhere サーバ - データベース管理』を参照してください)。

現在のクエリは、[結果] ウィンドウ枠の [結果] タブに表示される情報を生成した文です。現在のクエリがない場合、OUTPUT 文はエラーをレポートします。

INPUT 文は Interactive SQL コマンドなので、複合文 (IF など) やストアド・プロシージャには使用できません。「プロシージャ、トリガ、イベント、バッチで使用できる文」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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のみが表示されます。以前のクエリの結果は、すべて現在のクエリの結果に置き換えられます。

参照

- 「SELECT 文」 669 ページ
- 「INPUT 文 [Interactive SQL]」 585 ページ
- 「UNLOAD TABLE 文」 725 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

ASCII フォーマットのファイル中にある、Employee テーブルの内容を出力します。

```
SELECT *
FROM Employees;
OUTPUT TO Employees.txt
FORMAT ASCII;
```

Employee テーブルの内容を既存のファイルの終わりに置き、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。次の文を、HEXADECIMAL を ON に設定して実行します。

```
SELECT 'line1¥x0aline2';
OUTPUT TO file.txt HEXADECIMAL ON;
```

これにより、次のテキストを含む 1 行のファイルができます。

```
line10x0aline2
```

同じ文を、HEXADEMICAL 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 進数表現に変換されず、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。

PARAMETERS 文 [Interactive SQL]

この文は、Interactive SQL コマンド・ファイルにパラメータを指定するために使用します。

構文

PARAMETERS *parameter1*, *parameter2*, ⋯

備考

PARAMETERS コマンド・ファイルのパラメータに名前を付けて、コマンド・ファイルの中で後から参照できるようにします。

パラメータを参照するには、指定したパラメータ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名の間には、スペースを入れないでください。

コマンド・ファイルを呼び出すときに、すべての必要なパラメータを指定しないと、Interactive SQL は不足しているパラメータ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]」 637 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の Interactive SQL コマンド・ファイルは、2 つのパラメータを取ります。

```
PARAMETERS department_id, file;
SELECT Surname
FROM Employees
WHERE DepartmentID = { department_id }
>#{file}.dat;
```

このスクリプトを *test.sql* というファイルに保存すると、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]
```


PASSTHROUGH 文 [SQL Remote]

この文は、SQL Remote 管理のパススルー・モードの起動または停止に使用します。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に送られるだけで、サーバ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・プロシージャは、パススルー・コマンドを発行するサーバ上に存在していなければなりません (これらのストアド・プロシージャの中には、サーバ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 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。

パーミッション

DBA 権限が必要です。

関連する動作

なし。

例

```
PASSTHROUGH FOR rem_db ;
...
( SQL statements to be executed at the remote database )
...
PASSTHROUGH STOP ;
```

PREPARE 文 [ESQL]

この文は、後から実行する文、またはカーソルの定義に使用する文を準備するために使用します。

構文

PREPARE *statement-name*
  FROM *statement*
[**DESCRIBE** *describe-type* **INTO** [[**SQL**] **DESCRIPTOR**] *descriptor*]
[**WITH EXECUTE**]

statement-name : *identifier* or *hostvar*

statement : *string* or *hostvar*

describe-type :
 [**ALL** | **BIND VARIABLES** | **INPUT** | **OUTPUT** | **SELECT LIST**]
 [**LONG NAMES** [[[**OWNER.**]**TABLE.**]**COLUMN**]
     | **WITH VARIABLE RESULT**]

パラメータ

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 ID 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。

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・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。

静的と動的
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 statement name を参照し、文を実行します。または、statement が SELECT 文の場合、カーソルを開きます。*statement-name* は、*sqlca.h* ヘッダ・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、a_sql_statement_number というこの型の型定義があります。この型を SQL プリプロセッサが認識し、DECLARE セクションの中で使用できます。PREPARE 文の実行中に、データベースによってホスト変数が埋められるので、プログラマが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]」 489 ページ
- 「DESCRIBE 文 [ESQL]」 503 ページ
- 「OPEN 文 [ESQL] [SP]」 620 ページ
- 「EXECUTE 文 [ESQL]」 532 ページ
- 「DROP 文」 512 ページ

標準と互換性

- **SQL/2003** コア機能。

例

次の文は、簡単なクエリを準備します。

```
EXEC SQL PREPARE employee_statement FROM
'SELECT Surname FROM Employees';
```

PREPARE TO COMMIT 文

この文は、COMMIT を実行できるかどうかをチェックするために使用します。

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。

PREPARE TO COMMIT 文は、ストアド・プロシージャ、トリガ、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」 372 ページ
- 「ROLLBACK 文」 663 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文のシーケンスは、外部キーが Employee 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]

この文は、クライアントのウィンドウにメッセージを返すかデータベース・サーバの [メッセージ] ウィンドウにメッセージを表示するために使用します。

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 [サーバ・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を含むことができます。プレースホルダのフォームは *%nn!* で、*nn* は 1 ー 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」 616 ページ

標準と互換性

- **SQL/2003** 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]

この文は、指定したカーソルにローを挿入するために使用します。

構文

PUT *cursor-name*
[**USING DESCRIPTOR** *sqlda-name* | **FROM** *hostvar-list*]
[**INTO** { **DESCRIPTOR** *sqlda-name* | *hostvar-list* }]
[**ARRAY** *:nnn*]

cursor-name : *identifier* or *hostvar*

sqlda-name : *identifier*

hostvar-list : may contain indicator variables

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。あるいは、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カラムの値を取得します。

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 ポインタが NULL ポインタである場合、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。nnn は、挿入するローの数です。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。

カーソルへの挿入
スクロール (values 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。

データベースに LONG VARCHAR または LONG BINARY 値を入れる場合の詳細については、「SET 文」 677 ページを参照してください。

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。「カーソルによるローの変更」『SQL Anywhere サーバ - プログラミング』を参照してください。

参照

- 「UPDATE 文」 728 ページ
- 「UPDATE (位置付け) 文 [ESQL] [SP]」 733 ページ
- 「DELETE 文」 497 ページ
- 「DELETE (位置付け) 文 [ESQL] [SP]」 501 ページ
- 「INSERT 文」 590 ページ

標準と互換性

- **SQL/2003**　コア機能。

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 [T-SQL]

この文は、エラー信号を送り、クライアントにメッセージを送信するために使用します。

構文

RAISERROR *error-number* [*format-string*] [, *arg-list*]

パラメータ

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 テーブルから取得します。

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を含むことができます。プレースホルダのフォームは %*nn*! で、*nn* は 1 ー 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。

参照

- 「CREATE TRIGGER 文 [T-SQL]」 479 ページ
- 「on_tsql_error オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』
- 「continue_after_raiserror オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

例

次の文はエラー 23000 を発します。これはユーザ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 *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
```

接続を禁止する代替方法については、「login_procedure オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

READ 文 [Interactive SQL]

この文は、Interactive SQL 文をファイルから読み込むために使用します。

構文

READ [**ENCODING** *encoding*] *file-name* **[** *parameter* **]** ...

encoding : *identifier* or *string*

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。ファイル名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。Interactive SQL は、最初に現在のディレクトリを検索し、次に環境変数 SQLPATH の中で指定されたディレクトリを検索し、さらに環境変数 PATH の中で指定されたディレクトリを検索します。指定したファイルにファイル拡張子がない場合、Interactive SQL は各ディレクトリで拡張子 *.sql* を持つ同じファイル名を検索します。

encoding 引数では、ファイルの読み込みに使用される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。

encoding が指定されていない場合、Interactive SQL は、次のようにリストで先に出現するエンコード値を後で出現するエンコード値より優先することで、ファイルの読み込みに使用するエンコードを判断します。

- 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)
- Interactive SQL の開始時に -codepage で指定されたエンコード
- Interactive SQL が動作しているコンピュータのデフォルトのエンコード

Interactive SQL とエンコードの詳細については、「default_isql_encoding オプション [Interactive SQL]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

パラメータは、コマンド・ファイル名の後にリストできます。これらのパラメータは、文ファイルの先頭の PARAMETERS 文で指定したパラメータと対応します (「PARAMETERS 文 [Interactive SQL]」 627 ページを参照してください)。パラメータ名は角かっこ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を置き換えます。この *parameter-name* は適切なパラメータ名です。

コマンド・ファイルに渡すパラメータは、識別子、数、引用符付きの識別子、または文字列です。パラメータ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は、角カッコ ([ ]) で囲みます。こうすると、コマンド・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。

十分なパラメータがコマンド・ファイルに渡されない場合、Interactive SQL は足りないパラメータ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

- 「PARAMETERS 文 [Interactive SQL]」 627 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次は、READ 文の例です。

```
READ status.rpt '160'
READ birthday.sql [>= '1988-1-1'] [<= '1988-1-30']
```

READTEXT 文 [T-SQL]

この文は、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出すために使用します。

構文

READTEXT *table-name.column-name*
text-pointer offset size
[**HOLDLOCK**]

備考

READTEXT は、データベースからイメージ値とテキスト値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]」 748 ページ
- 「GET DATA 文 [ESQL]」 559 ページ
- 「TEXTPTR 関数 [テキストとイメージ]」 267 ページ

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

REFRESH MATERIALIZED VIEW 文

クエリ定義を実行することで、実体化ビュー (Materialized View) のデータを初期化または更新します。

構文

REFRESH MATERIALIZED VIEW [*owner*.]*materialized-view-name*
[**WITH** { **ISOLATION LEVEL** *isolation-level* | **EXCLUSIVE MODE** }]
[**FORCE BUILD**]

isolation-level :
0 | **1** | **2** | **3** | **snapshot** | **statement-snapshot** | **readonly-statement-snapshot**

パラメータ

WITH ISOLATION LEVEL isolation-level 句 この句は、更新操作を実行する場合の独立性レベルを変更するときに使用します。独立性レベルの詳細については、「トランザクションと独立性レベル」『SQL Anywhere サーバ - SQL の使用法』と「独立性レベルと一貫性」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

WITH EXCLUSIVE MODE 句 この句は、独立性レベルを変更しないが、基礎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したい場合に使用します。WITH EXCLUSIVE MODE を使用すると、実体化ビュー (Materialized View) から参照されているすべてのテーブルにテーブルのロックがかけられます。その結果、実体化ビュー (Materialized View) が更新されてもロックされているテーブルのデータは更新されなくなります。ただし、他の接続で基礎となるテーブルからデータを読み取ることはできます。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。

FORCE BUILD 句 デフォルトで、REFRESH MATERIALIZED VIEW 文を実行すると、データベース・サーバは基礎となるデータが変更されたかどうかをチェックします。変更されていない場合、実体化ビュー (Materialized View) は更新されません。FORCE BUILD を指定すると、基礎となるデータが変更されたかどうかに関係なく、実体化ビュー (Materialized View) が更新されます。

備考

この文は、実体化ビュー (Materialized View) を初期するときや、実体化ビュー (Materialized View) のデータを更新するときに使用します。更新とは、データベースがビューのクエリ定義を再実行し、返された新規データで実体化ビュー (Materialized View) のデータを置換し、結果として実体化ビュー (Materialized View) のデータが基礎となるテーブルのデータと一致する処理のことです。デフォルトで、データベース・サーバは、接続に設定されている現在の独立性レベルを使用して実体化ビュー (Materialized View) を更新します。

実体化ビュー (Materialized View) を更新する場合、および最適化にビューを使用する場合、一部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。さらに、各実体化ビュー (Materialized View) で記憶するオプションがあります。このようなオプションは、ビューを更新する場合、または最適化にビューを使用する場合、現在のオプションに一致する必要があります。特定の設定が必要なオプションの詳細については、「実体化ビュー (Materialized View) を管理するときの制限」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

パーミッション

実体化ビュー (Materialized View) には INSERT パーミッションが必要です。実体化ビュー (Materialized View) 定義のテーブルには SELECT パーミッションが必要です。

スナップショットのトランザクション内ではサポートされません。「スナップショット・アイソレーション」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

関連する動作

実体化ビュー (Materialized View) 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更新される実体化ビュー (Materialized View) に排他ロックがかけれます。また、実体化ビュー (Materialized View) から参照されるすべてのテーブルにブロックなしで共通のテーブル・ロックがかけられます。さらに、更新が完了するまで実体化ビュー (Materialized View) は初期化されていない状態にあるため、データベース・サーバまたはオプティマイザから使用できなくなります。

参照

- 「実体化ビュー (Materialized View) の編集」『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE MATERIALIZED VIEW 文」 420 ページ
- 「ALTER MATERIALIZED VIEW 文」 316 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、接続の独立性レベルを 1 (コミットされた読み込み) に変更し、ビューの再構築を強制することで、ProductIDsPerCustomer 実体化ビュー (Materialized View) のデータを更新します。

```
REFRESH MATERIALIZED VIEW ProductIDsPerCustomer
WITH ISOLATION LEVEL 1
FORCE BUILD;
```

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。

REFRESH TRACING LEVEL 文

REFRESH TRACING LEVEL 文は、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・レベルをリロードするときに使用します。

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。

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・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;
```

sa_diagnostic_tracing_level システム・テーブルの詳細については、「sa_diagnostic_tracing_level テーブル」 775 ページを参照してください。

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

- 「ATTACH TRACING 文」 348 ページ
- 「DETACH TRACING 文」 510 ページ
- 「診断トレーシングを使用した詳細なアプリケーション・プロファイリング」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

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 ユーザに制限し、そのユーザ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

この文は、現在のトランザクションで、セーブポイントを解放するために使用します。

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]」 359 ページ
- 「COMMIT 文」 372 ページ
- 「ROLLBACK 文」 663 ページ
- 「ROLLBACK TO SAVEPOINT 文」 664 ページ
- 「SAVEPOINT 文」 668 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 - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

REMOTE RESET 文 [SQL Remote]

この文は、カスタム・データベース抽出プロシージャで、単一トランザクションでの 1 リモート・ユーザのすべてのサブスクリプションの起動に使用します。

構文

REMOTE RESET *userid*

備考

このコマンドは、リモート・ユーザのサブスクリプションをすべて、1 つのトランザクション内で開始します。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、ISYSREMOTEUSER で作成し、起動した値を、このリモート・ユーザ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。

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 権限が必要です。

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]」 703 ページ
- 「ISYSREMOTEUSER システム・テーブル」 758 ページ

例

- 次の文は、リモート・ユーザ SamS のサブスクリプションをリセットします。

```
REMOTE RESET SamS;
```


REMOVE JAVA 文

この文は、クラスまたは JAR ファイルをデータベースから削除するために使用します。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。

クラスまたは jar は事前にインストールしておきます。

構文

REMOVE JAVA *classes-to-remove*

classes-to-remove :
CLASS *java-class-name*, … | **JAR** *jar-name*, …

パラメータ

CLASS *java-class-name* パラメータは、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 *jar-name* は、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。

パーミッション

DBA 権限が必要です。

Windows CE ではサポートされません。

標準と互換性

- **SQL/2003** ベンダ拡張。

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。

```
REMOVE JAVA CLASS Demo;
```

REORGANIZE TABLE 文

この文は、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するために使用します。

構文

REORGANIZE TABLE [*owner.*]*table-name*
[{ **PRIMARY KEY**
| **FOREIGN KEY** *foreign-key-name*
| **INDEX** *index-name* }
| **ORDER** {**ON** | **OFF**}
]

パラメータ

PRIMARY KEY　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　指定の外部キーを再編成します。

INDEX　指定のインデックスを再編成します。

ORDER オプション　ORDER を ON (デフォルト) に設定すると、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、データはプライマリ・キー値で順序付けられます。ORDER が OFF の場合、データはプライマリ・キーの順に配列されます。

クラスタード・インデックスの詳細については、「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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。各ローはプライマリ・キー (存在する場合) またはクラスタード・インデックスの順に処理されます。テーブルにプライマリ・キーまたはクラスタード・インデックスがない場合は、エラーが発生します。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。

どちらのフォームの再編成でも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。

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 CE ではサポートされません。
- スナップショットのトランザクション内ではサポートされません。「スナップショット・アイソレーション」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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。

例

次の文は、Employee 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、Employee 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、Product 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;
```

次の文は、Employee 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

この文は、例外条件を送り返すために使用します。

構文

RESIGNAL [*exception-name*]

備考

例外ハンドラの中で 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。RESIGNAL の前の例外ハンドラによる動作は取り消されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「SIGNAL 文」 696 ページ
- 「BEGIN 文」 356 ページ
- 「プロシージャとトリガでの例外ハンドラの使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「RAISERROR 文 [T-SQL]」 635 ページ

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

例

次のフラグメントはカラムが見つからない場合を除く、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

この文は、アーカイブからバックアップされたデータベースをリストアするために使用します。

構文

RESTORE DATABASE *file-name*
FROM *archive-root*
[**CATALOG ONLY**
| [**RENAME** *dbspace-name* **TO** *new-dbspace-name*] …]
[**HISTORY** { **ON** | **OFF** }]

file-name : *string* | *variable*
archive-root : *string* | *variable*
new-dbspace-name : *string* | *variable*

パラメータ

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 各 DB 領域をリストアする新しいロケーションを指定します。

HISTORY 句 デフォルトでは、各 RESTORE DATABASE 操作は *backup.syb* 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。次のすべての条件があてはまる場合は、ファイルが更新されないようにしたい場合があります。

- 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がない
- ディスク領域が非常に限られている

備考

各 RESTORE DATABASE 操作では、履歴ファイル *backup.syb* が更新されます。

backup.syb ファイルの詳細については、「SALOGDIR 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

RENAME 句によって、各 DB 領域のリストア・ロケーションを変更できます。RENAME 句には、DB 領域名として SYSTEM と TRANSLOG は指定できません。「事前定義の DB 領域」『SQL Anywhere サーバ - データベース管理』を参照してください。

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。

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・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。「-gu サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

この文は、Windows CE ではサポートされません。

関連する動作

なし。

参照

- 「BACKUP 文」 350 ページ
- 「バックアップとデータ・リカバリ」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。
- **Windows CE** Windows CE 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

この文は、結果セットを返すカーソルの実行を再開するために使用します。

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の実行を再開します。プロシージャは、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]」 489 ページ
- 「FETCH 文 [ESQL] [SP]」 543 ページ
- 「プロシージャから返される結果」 『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** ベンダ拡張。

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

この文は、関数、プロシージャ、またはバッチを無条件で終了し、オプションで値を返すために使用します。

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の値として返します。

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」 408 ページ
- 「CREATE PROCEDURE 文」 423 ページ
- 「BEGIN 文」 356 ページ

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

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 文

この文は、ユーザのパーミッションを取り消すために使用します。

構文 1

REVOKE *permission*, … **FROM** *userid*, …

permission :
CONNECT
| **DBA**
| **BACKUP**
| **VALIDATE**
| **INTEGRATED LOGIN**
| **KERBEROS LOGIN**
| **GROUP**
| **MEMBERSHIP IN GROUP** *userid*, …
| **RESOURCE**

構文 2

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*, …

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・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、プロシージャの EXECUTE パーミッションを取り消します。

REVOKE CONNECT はデータベースからユーザ ID を削除すると一緒に、そのユーザが所有しているオブジェクト (テーブル、ビュー、プロシージャなど) と、そのユーザ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が、別のユーザ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から MEMBERSHIP IN GROUP を自動的に取り消します。

グループにユーザを追加すると、ユーザ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、ユーザがグループのメンバ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 ID に割り当てる必要があります。

あるユーザに付与した WITH GRANT OPTION パーミッションを取り消すと、それ以前にそのユーザが他のユーザに付与した WITH GRANT OPTION パーミッションも取り消されます。

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」 565 ページ

標準と互換性

- **SQL/2003** 構文 1 はベンダ拡張です。構文 2 はコア機能です。構文 3 は永続的ストアド・モジュール機能です。

例

ユーザ Dave が Employee 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 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・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 ShowCustomers を実行できないようにします。


```
REVOKE EXECUTE ON ShowCustomers FROM Finance;
```

ユーザ ID FranW をデータベースから削除します。

```
REVOKE CONNECT FROM FranW;
```

REVOKE CONSOLIDATE 文 [SQL Remote]

この文は、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 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・リストから統合データベースのユーザ ID を削除します。

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。指定したユーザ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]」 659 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」 661 ページ
- 「REVOKE REMOTE DBA 文 [SQL Remote]」 662 ページ
- 「GRANT CONSOLIDATE 文 [SQL Remote]」 570 ページ

例

- 次の文は、ユーザ 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]

この文は、名前を指定したユーザ ID を CURRENT パブリッシャ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のユーザ ID によって識別されます。現在のパブリッシャのユーザ ID は、特殊定数 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 ID がパブリッシャ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・ユーザ 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・ユーザ 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・ユーザ 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 権限が必要です。

関連する動作

オートコミット。

参照

- 「GRANT PUBLISH 文 [SQL Remote]」 572 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」 661 ページ
- 「REVOKE REMOTE DBA 文 [SQL Remote]」 662 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」 658 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

```
REVOKE PUBLISH FROM publisher_ID;
```

REVOKE REMOTE 文 [SQL Remote]

この文は、このデータベースからの SQL Remote メッセージをユーザが受信できないようにします。

構文

REVOKE REMOTE FROM *userid*, …

備考

ユーザ 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・リストからユーザ ID を削除します。

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。指定したユーザ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]」 659 ページ
- 「GRANT REMOTE 文 [SQL Remote]」 573 ページ
- 「REVOKE REMOTE DBA 文 [SQL Remote]」 662 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」 658 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

```
REVOKE REMOTE FROM SamS;
```


REVOKE REMOTE DBA 文 [SQL Remote]

この文は、ユーザ ID に対して DBA 権限を取り消すために使用します。ただし、Message Agent から接続したときのみ有効です。

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 ID パスワードを送信する際のセキュリティを確保しながら、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。

- この文は、ユーザ ID から REMOTE DBA 権限を取り消します。

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

- 「REVOKE PUBLISH 文 [SQL Remote]」 659 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」 661 ページ
- 「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」 575 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」 658 ページ
- 「同期の開始」 『Mobile Link - クライアント管理』
- 「Message Agent とレプリケーション・セキュリティ」 『SQL Remote』

標準と互換性

- **SQL/2003** ベンダ拡張。

ROLLBACK 文

この文を使用して、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。

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」 372 ページ
- 「ROLLBACK TO SAVEPOINT 文」 664 ページ

標準と互換性

- **SQL/2003**　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]」 359 ページ
- 「COMMIT 文」 372 ページ
- 「RELEASE SAVEPOINT 文」 644 ページ
- 「ROLLBACK 文」 663 ページ
- 「SAVEPOINT 文」 668 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 - SQL の使用法』

標準と互換性

- **SQL/2003** コア SQL に含まれない SQL／基本機能。

ROLLBACK TRANSACTION 文 [T-SQL]

この文は、SAVE TRANSACTION の後に加えられた変更をキャンセルするために使用します。

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」 664 ページ
- 「BEGIN TRANSACTION 文 [T-SQL]」 359 ページ
- 「COMMIT 文」 372 ページ,
- 「SAVE TRANSACTION 文 [T-SQL]」 667 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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
```


ROLLBACK TRIGGER 文

この文は、トリガによって加えられた変更を元に戻すために使用します。

構文

ROLLBACK TRIGGER [**WITH** *raiserror-statement*]

備考

ROLLBACK TRIGGER 文は、トリガを起動するデータの修正など、トリガの中で実行された処理を元に戻します。

必要に応じて、RAISERROR 文を発行できます。RAISERROR 文を発行すると、エラーがアプリケーションに返されます。RAISERROR 文を発行しないと、エラーは返されません。

ネストしたトリガの中で ROLLBACK TRIGGER 文を使用し、RAISERROR 文を発行しないと、最も内側のトリガとそのトリガ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」 473 ページ
- 「ROLLBACK 文」 663 ページ
- 「ROLLBACK TO SAVEPOINT 文」 664 ページ
- 「RAISERROR 文 [T-SQL]」 635 ページ

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

SAVE TRANSACTION 文 [T-SQL]

この文は、現在のトランザクションで、セーブポイントを確立するために使用します。

構文

SAVE TRANSACTION *savepoint-name*

備考

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。*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。「トランザクション内のセーブポイント」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「SAVEPOINT 文」 668 ページ
- 「BEGIN TRANSACTION 文 [T-SQL]」 359 ページ
- 「COMMIT 文」 372 ページ
- 「ROLLBACK TRANSACTION 文 [T-SQL]」 665 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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
```


SAVEPOINT 文

この文は、現在のトランザクションで、セーブポイントを確立するために使用します。

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。「トランザクション内のセーブポイント」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

トリガまたはアトミック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」 644 ページ
- 「ROLLBACK TO SAVEPOINT 文」 664 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL／基本機能。

SELECT 文

この文は、データベースから情報を取り出すために使用します。

構文

[**WITH** *temporary-views*]
  SELECT [**ALL** | **DISTINCT**] [*row-limitation*] *select-list*
[**INTO** { *hostvar-list* | *variable-list* | *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** { **UPDATE** [*cursor-concurrency*] | **READ ONLY** }]
[**FOR XML** *xml-mode*]
[**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 :
  FIRST | **TOP** *n* [**START AT** *m*]

select-list :
expression [[**AS**] *alias-name*], ⋯
| *
| *window-function* **OVER** { *window-name* | *window-spec* }
  [[**AS**] *alias-name*]

from-expression :

「FROM 句」 552 ページを参照してください。

group-by-expression :

「GROUP BY 句」 576 ページを参照してください。

search-condition :

「探索条件」 20 ページを参照してください。

window-name : *identifier*

window-expression :

「WINDOW 句」 745 ページを参照してください。

window-spec :

「WINDOW 句」 745 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*

cursor-concurrency :
BY { **VALUES** | **TIMESTAMP** | **LOCK** }

xml-mode :
RAW [, **ELEMENTS**] | **AUTO** [, **ELEMENTS**] | **EXPLICIT**

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

パラメータ

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 文が次のいずれかの場所に表示される場合のみ、この句が使用できます。

- 最上位レベルの SELECT 文内
- VIEW 定義の最上位レベルの SELECT 文内
- INSERT 文内の最上位レベルの SELECT 文内

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。

「共通テーブル式」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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、文の実行に時間が非常に長くかかります。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 ORDER BY 句を含むクエリのローを明示的に制限します。TOP 値は 0 以上の整数定数または整数変数にします。START AT 値は 0 より大きい整数定数または整数変数にします。

FIRST と TOP の使用の詳細については、「クエリが返すロー数を明示的に制限する」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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 (「SQL 関数」 91 ページを参照してください)。サブクエリも、*select-list* で許可されます (「式」 15 ページを参照してください)。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 は式自体を表示します。

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を指定してください。*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・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。

INTO variable-list 句 この句はプロシージャとトリガ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 つの変数項目が必要です。*select-list* 項目は、順番に変数に配置されます。

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。

テーブル名が # から始まる場合は、テンポラリ・テーブルとして作成されます。それ以外の場合、テーブルは永久ベース・テーブルとして作成されます。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。文は、CREATE TABLE の後に INSERT ... SELECT が続く省略形です。

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。

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;
```

「FROM 句」 552 ページを参照してください。

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 句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。「探索条件」 20 ページと「FROM 句」 552 ページを参照してください。

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。例外は、*select-list* と HAVING 句が集合関数を持つ場合だけです。

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。「WINDOW 句」 745 ページを参照してください。

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、クエリに対してひら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。

FOR UPDATE BY TIMESTAMP または FOR UPDATE BY VALUES を指定すると、データベース・サーバは keyset-driven カーソルを使用して最適な同時実行性を使用します。この場合、実行されない更新が発生する可能性があります。

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 (「DECLARE 文」 488 ページと「FOR 文」 547 ページを参照してください)。ODBC、JDBC、OLE DB では、文の更新可能性は明示的で読み込み専用です。アプリケーションが上書きしなければ、forward-only カーソルが使用されます。Open Client、Embedded SQL、ストアド・プロシージャでは、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。

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

カーソルの更新可能性とは別に、文の更新可能性も 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。

カーソルの sensitivity の詳細については、「SQL Anywhere のカーソル」 『SQL Anywhere サーバ - プログラミング』を参照してください。

ODBC の同時性の詳細については、「ODBC カーソル特性の選択」 『SQL Anywhere サーバ - プログラミング』の SQLSetStmtAttr の説明を参照してください。

ansi_update_constraints データベース・オプションの詳細については、「ansi_update_constraints オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

カーソルの updatability の詳細については、「更新可能な文の概要」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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。

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 ドキュメントのフォームを制御できます。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。「FOR XML EXPLICIT の使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

FOR XML 句の使用の詳細については、「FOR XML 句を使用してクエリ結果を XML として取り出す」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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 '*option-value*'** MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザがクエリを処理するときに実体化ビュー (Materialized View) 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。「materialized_view_optimization オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。
- **FORCE OPTIMIZATION** 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省略されます。ただし、コストベースの最適化を実行したい場合もあります。たとえば、クエリ処理時に実体化ビュー (Materialized View) 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、ビューのマッチング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、プロシージャ内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャに対するすべての呼び出しにオプティマイザ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。

  単純なクエリとビューのマッチングに関する詳細については、「クエリ処理のフェーズ」 『SQL Anywhere サーバ - SQL の使用法』と「実体化ビュー (Materialized View) によるパフォーマンスの向上」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- ***option-name = option-value*** 該当する文に対してのみ、パブリック・オプションやテンポラリ・オプションの設定よりも優先されるオプション設定を指定します。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 [互換性]」 『SQL Anywhere サーバ - データベース管理』

- 「max_query_tasks オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

備考

SELECT 文を使用して、データベースからの結果を取り出します。

SELECT 文を Interactive SQL 内で使用して、データベース内のデータをブラウズしたり、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できます。

SELECT 文は、プロシージャとトリガ、または ESQL 内でも使用できます。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。

SELECT 文を使って、プロシージャから結果セットを返すこともできます。

> **注意**
> SELECT 文で GROUP BY 式を使う場合、*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。例外は、*select-list* と HAVING 句が集合関数を持つ場合だけです。

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。

関連する動作

なし。

参照

- 「式」 15 ページ
- 「FROM 句」 552 ページ
- 「探索条件」 20 ページ
- 「UNION 文」 720 ページ
- 「ジョイン：複数テーブルからのデータ検索」『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　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、ROLLUP キーワードは T431 の一部です。

  FOR UPDATE、FOR READ ONLY、FOR UPDATE (*column-list*) はコア機能です。

  FOR UPDATE BY [LOCK | TIMESTAMP | VALUES] は SQL Anywhere のベンダ拡張です。

例

この例は、Employees テーブルの全従業員数を返します。

```
SELECT COUNT(*)
FROM Employees;
```

この例では、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。

```
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 文

この文は、SQL 変数に値を代入するために使用します。

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、CREATEVARIABLE 文または DECLARE 文を使って事前に作成されたものであるか、プロシージャの OUTPUT パラメータ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 32,767 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」 480 ページ
- 「DECLARE 文」 488 ページ
- 「DROP VARIABLE 文」 528 ページ
- 「式」 15 ページ

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

例

この単純な例では、「birthday」という変数の作成を表示し、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( 500 ) 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;
  /* 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]

この文は、Adaptive Server Enterprise と互換性を持つ方法で現在の接続にデータベース・オプションを設定するために使用します。

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。 週の最初の日を指定する方法の詳細については、「first_day_of_week オプション [データベース]」『SQL Anywhere サーバ - データベース管理』と「DATEPART 関数 [日付と時刻]」 140 ページを参照してください。
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。しかし、SQL Anywhere は、特に互換性に対して便利なオプションを持つ Adaptive Server Enterprise の SET 文もサポートします。

次のオプションは、Adaptive Server Enterprise 同様、SQL Anywhere の Transact-SQL SET 文を使って設定できます。

- **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 [互換性]」『SQL Anywhere サーバ - 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。「ansi_permissions オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **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。「close_on_endtrans オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **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。このオプションを永続的に設定する方法については、「first_day_of_week オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **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。「Transact-SQL との互換性を維持するためのオプション設定」『SQL Anywhere サーバ - SQL の使用法』と「quoted_identifier オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **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) 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
      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。したがって、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内で使用して、そのトリガに関連するテーブルの操作が他のトリガを起動できるか (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。「string_rtruncation オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。
- **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 | read only statement snapshot }** 「独立性レベルと一貫性」 『SQL Anywhere サーバ - SQL の使用法』で説明されている、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。「isolation_level オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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

- 「SET OPTION 文」 686 ページ

- 「Transact-SQL との互換性を維持するためのオプション設定」『SQL Anywhere サーバ - SQL の使用法』
- 「互換性オプション」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

SET CONNECTION statement [Interactive SQL] [ESQL]

この文は、アクティブなデータベース接続を変更するために使用します。

構文

SET CONNECTION [*connection-name*]

connection-name : *identifier*, *string*, or *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。

参照

- 「CONNECT 文 [ESQL] [Interactive SQL]」 375 ページ
- 「DISCONNECT 文 [ESQL] [Interactive SQL]」 511 ページ

標準と互換性

- **SQL/2003** Interactive SQL はベンダ拡張です。Embedded SQL はコア機能です。

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]

この文は、SQLDA 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。

構文

SET DESCRIPTOR *descriptor-name*
{ **COUNT** = { *integer* | *hostvar* }
| **VALUE** { *integer* | *hostvar* } *assignment*, ⋯ }

assignment :
{ **TYPE** | **SCALE** | **PRECISION** | **LENGTH** | **INDICATOR** }
= { *integer* | *hostvar* }
| **DATA** = *hostvar*

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 を実行すると型チェックが実行され、記述子領域内の変数とホスト変数の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]」 301 ページ
- 「DEALLOCATE DESCRIPTOR 文 [ESQL]」 486 ページ
- 「SQLDA (SQL descriptor area)」 『SQL Anywhere サーバ - プログラミング』

標準と互換性

- **SQL/2003** コア SQL に含まれない SQL／基本機能。

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;
  }
```

詳細例については、「ALLOCATE DESCRIPTOR 文 [ESQL]」 301 ページを参照してください。

SET OPTION 文

この文は、データベース・オプションの値を変更するために使用します。

構文

SET [**EXISTING**] [**TEMPORARY**] **OPTION**
[*userid*.| **PUBLIC**.]*option-name* = [*option-value*]

userid : *identifier*, *string*, or *hostvar*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

Embedded SQL 構文

SET [**TEMPORARY**] **OPTION**
[*userid*.| **PUBLIC**.]*option-name* = [*option-value*]

userid : *identifier*, *string*, or *hostvar*

option-name : *identifier*, *string*, or *hostvar*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

備考

SET OPTION 文を使用すると、データベース・サーバ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、すべてのユーザまたは個々のユーザだけの動作を変更できます。変更のスコープは、テンポラリまたは永続的のどちらかを指定できます。

どのようなオプションでも、ユーザ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・サーバでは、ユーザ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。

オプションのクラスは次のとおりです。

- 一般的なデータベース・オプション
- Transact-SQL との互換性
- 複写データベース・オプション

使用可能なすべてのオプションの一覧と説明については、「データベース・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

オプションは、パブリック、ユーザ、テンポラリの 3 つのスコープ・レベルで設定できます。テンポラリ・オプションは他のオプションより優先され、ユーザのオプションはパブリック・オプションより優先されます。現在のユーザに対してユーザ・レベルのオプションを設定すると、対応するテンポラリ・オプションも設定されます。

EXISTING キーワードを使用した場合、個別のユーザ ID のオプション値は、そのオプションにすでに PUBLIC ユーザ ID 設定が行われていないかぎり、設定できません。

ユーザ ID を指定すると、オプション値がそのユーザ (またはグループのメンバであるグループ・ユーザ ID) に適用されます。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 文を発行する、現在ログオンしているユーザ ID に適用されます。

たとえば、次の文は、DBA が SQL 文を発行しているユーザである場合、ユーザ DBA にオプションの変更を適用します。

```
SET OPTION precision = 40;
```

しかし、次の文は、すべてのユーザが属するユーザ・グループである PUBLIC ユーザ ID に変更を適用します。

```
SET OPTION Public.login_mode = Standard;
```

DBA 権限を持つユーザのみ、PUBLIC ユーザ ID にオプションを設定する権限を持ちます。

SET OPTION 文を使用すると、自分のユーザ ID の値を変更できます。他のユーザのユーザ ID のオプション値を設定できるのは、DBA 権限を持っている場合だけです。

SET OPTION 文に TEMPORARY キーワードを追加すると、変更の効果の継続期間を変更できます。デフォルトでは、オプション値は永久です。SET OPTION 文を使って明示的に変更されるまで変わりません。

SET TEMPORARY OPTION 文がユーザ 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 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 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。

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。個人的なオプション設定である場合は、値は PUBLIC 設定に戻されます。TEMPORARY オプションを削除すると、オプション設定は永久設定に戻されます。

警告
カーソルからローをフェッチしている間のオプション設定変更は、動作の定義を曖昧にすることになるため、サポートされていません。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。ローをフェッチしている間にオプション設定を変更しないでください。

SET OPTION 文は、SQL FLAGGER で無視されます。

パーミッション

ユーザ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 - データベース管理』
- 「互換性オプション」『SQL Anywhere サーバ - データベース管理』
- 「SQL Remote オプション」『SQL Anywhere サーバ - データベース管理』
- 「SET OPTION 文 [Interactive SQL]」 689 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

日付フォーマット・オプションをオンに設定します。

```
SET OPTION public.date_format = 'Mmm dd yyyy';
```

日付フォーマット・オプションをデフォルト設定に設定します。

```
SET OPTION public.date_format =;
```

wait_for_commit オプションを On に設定します。

```
SET OPTION wait_for_commit = 'On';
```

次に ESQL の例を 2 つ示します。

```
1. EXEC SQL SET OPTION :user.:option_name = :value;
2. EXEC SQL SET TEMPORARY OPTION date_format = 'mm/dd/yyyy';
```


SET OPTION 文 [Interactive SQL]

この文は、Interactive SQL オプションの値を変更するために使用します。

構文 1

SET [**TEMPORARY**] **OPTION** *option-name* = [*option-value*]

userid : *identifier*, *string*, or *hostvar*

option-name : *identifier*, *string*, or *hostvar*

option-value : *string*, *identifier*, or *number*

構文 2

SET PERMANENT

構文 3

SET

備考

構文 1 は、指定された Interactive SQL オプションを格納します。

構文 2 は、現在のすべての Interactive SQL オプションを格納します。

構文 3 は、現在のデータベース・オプション設定すべてを表示します。データベース・サーバにテンポラリ・オプションが設定されている場合、これらが表示されます。設定されていない場合は、永続的なオプション設定が表示されます。

Interactive SQL のオプション設定はクライアント・コンピュータに保存されます。これらのオプション設定はデータベースには保存されません。

注意
構文 SET [TEMPORARY] OPTION [*userid* | PUBLIC.]*option-name* は、廃止されました。この構文を指定すると、*userid* または PUBLIC キーワードは無視されます。

参照

- 「Interactive SQL オプション」 『SQL Anywhere サーバ - データベース管理』

SET REMOTE OPTION 文 [SQL Remote]

この文は、SQL Remote メッセージ・リンクのメッセージ制御パラメータの設定に使用します。

構文

SET REMOTE *link-name* **OPTION**
[*userid*.| **PUBLIC**.]*link-option-name* = *link-option-value*

link-name:
file | **ftp** | **mapi** | **smtp** | **vim**

link-option-name:
common-option | *file-option* | *ftp-option*
| *mapi-option* | *smtp-option* | *vim-option*

common-option:
debug | **output_log_send_on_error**
| **output_log_send_limit** | **output_log_send_now**

file-option:
directory | **invalid_extensions** | **unlink_delay**

ftp-option:
active_mode | **host** | **invalid_extensions** | **password** | **port** | **root_directory** | **user** | **reconnect_retries** | **reconnect_pause**

mapi-option:
force_download | **ipm_receive** | **ipm_send** | **profile**

smtp-option:
local_host | **pop3_host** | **pop3_password** | **pop3_userid**
| **smtp_host** | **top_supported**

vim-option:
password | **path** | **receive_all** | **send_vim_mail** | **userid**

link-option-value : *string*

パラメータ

userid *userid* を指定しない場合、現在のパブリッシャが使用されます。

オプション値 オプション値は、メッセージリンクによって異なります。詳細については、次の項を参照してください。

- 「FILE メッセージ・システム」『SQL Remote』
- 「FTP メッセージ・システム」『SQL Remote』
- 「MAPI メッセージ・システム」『SQL Remote』
- 「SMTP メッセージ・システム」『SQL Remote』
- 「VIM メッセージ・システム」『SQL Remote』

備考

メッセージ・リンクの初回使用時に、ユーザがメッセージ・リンク・パラメータをメッセージ・リンクのダイアログ・ボックスで入力すると、Message Agent はそのパラメータを保存します。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 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。

注意
VIM および MAPI のサポートは廃止されました。

パーミッション

DBA 権限が必要です。パブリッシャ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・サイトでのトラブルシューティング・エラー」 『SQL Remote』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、ユーザ 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';
```

SET SQLCA 文 [ESQL]

この文は、デフォルトのグローバル *sqlca* 以外の SQLCA を使うように、SQL プリプロセッサに通知するために使用します。

構文

SET SQLCA *sqlca*

sqlca : *identifier* or *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 プリプロセッサに通知します。sqlca は、SQLCA ポインタに対する 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 ポインタを暗黙のうちにデータベース・インタフェース・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だけです (「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 - プログラミング』を参照してください)。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 - プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

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 );
}
```


SETUSER 文

この文は、データベース管理者が別のユーザと同一化し、接続プールを有効にするために使用します。

構文

{ **SET SESSION AUTHORIZATION** | **SETUSER** }
[[**WITH OPTION**] *userid*]

パラメータ

WITH OPTION　デフォルトでは、パーミッション (グループ・メンバシップを含む) 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 ID は、識別子 (SETUSER 構文) または文字列 (SET SESSION AUTHORIZATION 構文) です。「識別子」 7 ページと「文字列」 8 ページを参照してください。

備考

SETUSER 文はデータベース管理者の仕事を簡単にするために用意されています。DBA 権限を持つユーザがデータベースの他のユーザと同一化することを可能にします。SETUSER 文を実行した後に、次のコマンドの 1 つを実行して、同一化しているユーザをチェックできます。

- SELECT USER
- SELECT CURRENT USER

また、アプリケーション・サーバから SETUSER を使って、接続プールを利用できます。接続プールによって、必要な個々の接続の数が減り、パフォーマンスが向上します。

ユーザ 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、トリガ、イベント・ハンドラ、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。

- **オブジェクトの作成**　SETUSER を使用して、別のユーザ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、別のユーザのパーミッションとグループ・メンバシップを使用し、クエリ、プロシージャ、ビューなどのパーミッションや名前解決をテストできます。
- **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 ID に切り替えることができます。

> **注意**
> プロシージャ、トリガ、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 権限が必要です。

参照

- 「EXECUTE IMMEDIATE 文 [SP]」 536 ページ
- 「GRANT 文」 565 ページ
- 「REVOKE 文」 655 ページ
- 「SET OPTION 文」 686 ページ

標準と互換性

- **SQL/2003** SET SESSION AUTHORIZATION はコア機能です。SETUSER はベンダ拡張です。

例

次の文は、ユーザ名 DBA によって実行され、ユーザ 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、ユーザを Jane に設定します。ユーザ 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

この文は、例外条件の信号を発信するために使用します。

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。例外の処理方法の説明については、「プロシージャとトリガでの例外ハンドラの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

exception-name　現在の複合文の始めにある DECLARE 文で宣言された例外の名前。この例外は、システム定義の SQLSTATE またはユーザ定義の SQLSTATE と合致していなければなりません。ユーザ定義 SQLSTATE の値は 99000 ー 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」 649 ページ
- 「BEGIN 文」 356 ページ
- 「プロシージャとトリガでの例外ハンドラの使用」『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003**　永続的ストアド・モジュール機能。

例

次の複合文は、ユーザ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、[メッセージ] タブにメッセージが表示されます。

```
BEGIN
  DECLARE myexception EXCEPTION
  FOR SQLSTATE '99001';
  SIGNAL myexception;
  EXCEPTION
    WHEN myexception THEN
      MESSAGE 'My exception signaled'
      TO CLIENT;
END
```

START DATABASE 文

この文は、現在のデータベース・サーバ上でデータベースを起動するために使用します。

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*]

パラメータ

start-options は任意の順序で表示できます。

START DATABASE 句　*database-file* パラメータは文字列です。*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・サーバの開始ディレクトリを基準にします。

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・サーバは、現在実行しているサーバのうち、最初に起動したサーバです。*server-name* パラメータは識別子です。

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　読み込み専用モードでデータベースを起動します。

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・サーバに接続するときにそのサーバの代替名を指定するときに使用します。データベースのミラーリングを使用している場合、プライマリ・サーバとミラー・サーバの両方に同じデータベース・サーバ名を付ける必要があります。これは、クライアントは接続先がどちらのサーバになるかわからないためです。

代替サーバ名とデータベースのミラーリングの詳細については、「-sn オプション」『SQL Anywhere サーバ - データベース管理』と「データベース・ミラーリングの概要」『SQL Anywhere サーバ - データベース管理』を参照してください。

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・サーバがすべての DB 領域と同じディレクトリで起動されている場合に DIRECTORY '.' 句を含めると、この句により、データベース・サーバ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。「-ds データベース・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

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。

ユーティリティ・データベースについては、「ユーティリティ・データベースの使用」『SQL Anywhere サーバ - データベース管理』を参照してください。

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・サーバ以外のサーバで起動できます。

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・サーバでは all、ネットワーク・サーバでは 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」 707 ページ
- 「CONNECT 文 [ESQL] [Interactive SQL]」 375 ページ
- 「-gd サーバ・オプション」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

現在のサーバで、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 sample 上で起動します。

```
START DATABASE 'c:¥database files¥sample_2.db'
AS sam2
ON sample;
```


START ENGINE 文 [Interactive SQL]

この文は、データベース・サーバを起動するために使用します。

構文

START ENGINE AS *database-server-name* [**STARTLINE** *command-string*]

備考

START ENGINE 文は、データベース・サーバを起動します。データベース・サーバ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。有効なコマンド文字列は、「SQL Anywhere データベース・サーバ」『SQL Anywhere サーバ - データベース管理』のデータベース・サーバの説明に従った文字列です。

パーミッション

なし。

関連する動作

なし。

参照

- 「STOP ENGINE 文」 708 ページ
- 「SQL Anywhere データベース・サーバ」『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** ベンダ拡張。

例

sample という名前のデータベース・サーバを、1 つもデータベースを起動しない状態で起動します。

```
START ENGINE AS sample;
```

次の例は、STARTLINE 句の使い方を示します。

```
START ENGINE AS eng1 STARTLINE 'dbeng10 -c 8M';
```

START JAVA 文

この文は、Java VM を起動するために使用します。

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。「java_location オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Java VM の起動の詳細については、「Java VM の起動と停止」 『SQL Anywhere サーバ - プログラミング』を参照してください。

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 CE ではサポートされません。

関連する動作

なし。

参照

- 「STOP JAVA 文」 709 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

Java VM を起動します。

```
START JAVA;
```


START LOGGING 文 [Interactive SQL]

この文は、実行した SQL 文のログ・ファイルへの記録を開始するために使用します。

構文

START LOGGING *file-name*

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。

パーミッション

なし。

関連する動作

なし。

参照

- 「STOP LOGGING 文 [Interactive SQL]」 710 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

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]

この文は、パブリケーションに対するユーザのサブスクリプションの開始に使用します。

構文

START SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*, …

パラメータ

publication-name このパブリケーション名でユーザ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のユーザ ID。このユーザは、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**」とみなされます。

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 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE SUBSCRIPTION 文 [SQL Remote]」 453 ページ
- 「REMOTE RESET 文 [SQL Remote]」 645 ページ
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」 714 ページ
- 「STOP SUBSCRIPTION 文 [SQL Remote]」 711 ページ
- 「データベース抽出ユーティリティ」 『SQL Remote』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、パブリケーション **pub_contact** に対するユーザ **SamS** のサブスクリプションを起動します。

```
START SUBSCRIPTION TO pub_contact
FOR SamS;
```

START SYNCHRONIZATION DELETE 文 [Mobile Link]

次の文を使用して 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 権限が必要です。

関連する動作

なし。

参照

- 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」 713 ページ
- 「StartSynchronizationDelete メソッド」 『Ultra Light - .NET プログラミング』

標準と互換性

- **SQL/2003** ベンダ拡張。

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;
```

STOP DATABASE 文

この文は、現在のデータベース・サーバ上のデータベースを停止するために使用します。

構文

STOP DATABASE *database-name*
[**ON** *database-server-name*]
[**UNCONDITIONALLY**]

パラメータ

STOP DATABASE 句　*database-name* は、現在のサーバで動作している (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・サーバで起動された場合のみ停止されます。

UNCONDITIONALLY キーワード　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・サーバの -gk 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・サーバでは all、ネットワーク・サーバでは 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。

参照

- 「START DATABASE 文」 697 ページ
- 「DISCONNECT 文 [ESQL] [Interactive SQL]」 511 ページ
- 「-gd サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

現在のサーバ上のデータベース *sample* を停止します。

```
STOP DATABASE sample;
```

STOP ENGINE 文

この文は、データベース・サーバを停止するために使用します。

構文

STOP ENGINE [*database-server-name*] [**UNCONDITIONALLY**]

パラメータ

STOP ENGINE 句　*database-server-name* は Interactive SQL でのみ使用できます。この文を Interactive SQL で実行しない場合は、現在のデータベース・サーバが停止されます。

UNCONDITIONALLY キーワード　停止するデータベース・サーバに接続しているユーザが他にいない場合、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・サーバが停止します。

備考

STOP ENGINE 文は、指定したデータベース・サーバを停止します。UNCONDITIONALLY キーワードを指定すると、データベース・サーバはサーバ上に他の接続がある場合でも停止します。デフォルトでは、データベース・サーバは他の接続があると停止しません。

STOP ENGINE 文は、ストアド・プロシージャ、トリガ、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

サーバ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・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・サーバのデフォルト設定は all で、ネットワーク・サーバのデフォルト設定は DBA です。

関連する動作

なし。

参照

- 「START ENGINE 文 [Interactive SQL]」 700 ページ
- 「-gk サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

現在のデータベース・サーバを他に接続がない場合のみ停止します。

```
STOP ENGINE;
```

STOP JAVA 文

この文は、Java VM を停止するために使用します。

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 CE ではサポートされません。

関連する動作

なし。

参照

- 「START JAVA 文」 701 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

Java VM を停止します。

```
STOP JAVA;
```


STOP LOGGING 文 [Interactive SQL]

この文は、現在のセッションで実行される SQL 文のロギングを停止するために使用します。

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。

パーミッション

なし。

関連する動作

なし。

参照

- 「START LOGGING 文 [Interactive SQL]」 702 ページ

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 SUBSCRIPTION 文 [SQL Remote]

この文は、パブリケーションに対するユーザ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

publication-name　このパブリケーション名でユーザ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のユーザ ID。このユーザは、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**」とみなされます。

サブスクライバ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 権限が必要です。

関連する動作

オートコミット。

参照

- 「DROP SUBSCRIPTION 文 [SQL Remote]」 524 ページ
- 「START SUBSCRIPTION 文 [SQL Remote]」 703 ページ
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」 714 ページ
- 「データベース抽出ユーティリティ」『SQL Remote』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、パブリケーション **pub_contact** に対するユーザ **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]

次の文を使用して 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、同期に組み込まれ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。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。

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。

このコマンドは、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 権限が必要です。

関連する動作

なし。

参照

- Ultra Light 「StartSynchronizationDelete メソッド」 『Ultra Light - .NET プログラミング』
- Ultra Light 「StopSynchronizationDelete メソッド」 『Ultra Light - .NET プログラミング』
- 「START SYNCHRONIZATION DELETE 文 [Mobile Link]」 705 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

例については、「START SYNCHRONIZATION DELETE 文 [Mobile Link]」 705 ページを参照してください。

SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]

この文は、パブリケーションに対するユーザのサブスクリプションの同期に使用します。

構文

SYNCHRONIZE SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *remote-user*, ⋯

パラメータ

publication-name　このパブリケーション名でユーザ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のユーザ ID。このユーザは、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**」とみなされます。

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 権限が必要です。

関連する動作

オートコミット。

参照

- 「CREATE SUBSCRIPTION 文 [SQL Remote]」 453 ページ
- 「START SUBSCRIPTION 文 [SQL Remote]」 703 ページ
- 「STOP SUBSCRIPTION 文 [SQL Remote]」 711 ページ
- 「データベース抽出ユーティリティ」『SQL Remote』

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、パブリケーション **pub_contact** に対するユーザ **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]

この文を使用して、Interactive SQL から実行ファイルを起動します。

構文

SYSTEM *'* [*path*] *file-name '*

備考

指定した実行ファイルを起動します。

- SYSTEM 文はすべてを 1 行に記述してください。
- SYSTEM 文の終わりにコメントは記述できません。
- 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]」 375 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、メモ帳プログラムを起動します。メモ帳の実行ファイルが、起動可能なパスにあることが前提となります。

```
SYSTEM 'notepad.exe';
```

TRIGGER EVENT 文

この文は、指定したイベントをトリガするために使用します。イベントは、イベント・トリガ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

parm = value　トリガ条件によってイベント・ハンドラが実行されるときには、データベース・サーバが event_parameter 関数を使用して、イベント・ハンドラ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を明示的に指定し、イベント・ハンドラのコンテキストをシミュレートできます。

備考

トリガ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条件が発生していなくても、イベント・ハンドラ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・ハンドラを実行しません。

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

- 「ALTER EVENT 文」 311 ページ
- 「CREATE EVENT 文」 397 ページ
- 「EVENT_PARAMETER 関数 [システム]」 161 ページ

例

次の例は、イベントに文字列パラメータを渡す方法を示します。イベントは、トリガされた時刻をデータベース・サーバ・コンソールに表示します。

```
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' || event_parameter( 'time' );
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string( current timestamp ) );
```

TRUNCATE TABLE 文

この文は、テーブル定義は削除せずにテーブルからすべてのローを削除するために使用します。

構文

TRUNCATE TABLE [*owner*.]*table-name*

備考

TRUNCATE TABLE 文は、テーブルからすべてのローを削除します。これは WHERE 句のない DELETE 文と同じですが、TRUNCATE TABLE 文の結果としてトリガを起動しない点と、個々のロー削除をトランザクション・ログの中へ入れない点が異なります。

注意
同期またはレプリケーションが関係するデータベースにこの文を使用するときは注意が必要です。TRUNCATE TABLE 文は、テーブルからすべてのローを削除します。DELETE 文と似ていますが、WHERE 句を指定できません。ただし、TRUNCATE TABLE 文の結果としてトリガは発生しません。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 TABLE 文の後、テーブル構造体とインデックスのすべては、DROP TABLE 文が発行されるまで存在し続けます。カラム定義と制約はそのまま残り、トリガとパーミッションは有効なままです。

注意
TRUNCATE TABLE 文は、データ定義文のように 1 つの文としてトランザクション・ログに記録されます。それぞれの削除されたローは、トランザクション・ログには記録されません。TRUNCATE TABLE 文は dbmlsync、dbremote、dbltm からレプリケートまたは同期しません。DELETE 文のみがレプリケートまたは同期されます。

truncate_with_auto_commit オプションが On (デフォルト) に設定されている場合は、以下の条件がすべて満たされるときに、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は、そのオペレーションの前後に COMMIT が実行されます。高速トランケーションを実行する場合、各 DELETE はトランザクション・ログに記録されません。

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・テーブルの場合、TRUNCATE TABLE 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、事前に開かれている、トランケートされるテーブルを参照するカーソルを閉じ、COMMIT または ROLLBACK を発行して、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、各ユーザはデータの独自のコピーを所有しているため、排他的アクセスは不要です。

高速トランケーションを実行する場合、スナップショット・トランザクション内では TRUNCATE TABLE を使用できません。「スナップショット・アイソレーション」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

関連する動作

TRUNCATE TABLE 文は、トリガ削除を実行しません。

truncate_with_auto_commit を On に設定すると、テーブルがトランケートされる前後に COMMIT が実行されます。

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 TABLE 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、TRUNCATE TABLE 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」 497 ページ
- 「truncate_with_auto_commit オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

例

Department テーブルからすべてのローを削除します。

```
TRUNCATE TABLE Departments;
```


UNION 文

この文は、2 つ以上の SELECT 文の結果を結合するために使用します。

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-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

パラメータ

注意
query-block で FOR、FOR XML、WITH または OPTION の句を使用することはできません。

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 '*option-value*'** MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザがクエリを処理するときに実体化ビュー (Materialized View) 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。「materialized_view_optimization オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

- **FORCE OPTIMIZATION** 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省略されます。ただし、コストベースの最適化を実行したい場合もあります。たとえば、クエリ処理時に実体化ビュー (Materialized View) 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、ビューのマッチング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、プロシージャ内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャに対するすべての呼び出しにオプティマイザ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。

単純なクエリとビューのマッチングに関する詳細については、「クエリ処理のフェーズ」『SQL Anywhere サーバ - SQL の使用法』と「実体化ビュー (Materialized View) によるパフォーマンスの向上」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- ***option-name = option-value*** 該当する文に対してのみ、パブリック・オプションやテンポラリ・オプションの設定よりも優先されるオプション設定を指定します。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 [互換性]」『SQL Anywhere サーバ - データベース管理』
  - 「max_query_tasks オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

備考

いくつかのクエリ・ブロックの結果を、UNION を使ってより大きな結果へと結合することができます。各 *query-block* では、それぞれ select リストの中の項目数が同じになるようにしてください。

UNION ALL の結果は、単にクエリ・ブロックの結果を結合したものです。UNION の結果は UNION ALL と同じですが、重複ローが削除されている点が異なります。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 select リスト中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。

UNION の最初のクエリ・ブロックは、ORDER BY 句と一致する名前を判断するために使用します。

表示されるカラム名は、最初の *query-block* に対して表示されるカラム名と同じです。結果セットのカラム名をカスタマイズするには、*query-block* で WITH 句を使用するという方法もあります。

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

- 「SELECT 文」 669 ページ

標準と互換性

- **SQL/2003** コア機能。

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

この文は、データベースから外部 ASCII フォーマット・ファイルにデータをエクスポートするために使用します。

構文

UNLOAD *select-statement*
TO *file-name*
[*unload-option* …]

unload-option :
APPEND {**ON**|**OFF**}
| **DELIMITED BY** *string*
| **ESCAPE CHARACTER** *character*
| **ESCAPES** {**ON** | **OFF**}
| **FORMAT** {**ASCII** | **BCP**}
| **HEXADECIMAL** {**ON** | **OFF**}
| **QUOTE** *string*
| **QUOTES** {**ON** | **OFF**}

file-name : *string* | *variable*

パラメータ

file-name　データのアンロード先となるファイルの名前。文を実行するのはデータベース・サーバであるため、*file-name* はデータベース・サーバ・コンピュータ上のファイルを示します。相対の場合の *file-name* は、データベース・サーバの開始ディレクトリを基準に相対パスでファイルを示します。クライアント・コンピュータへのデータのアンロードについては、「PASSTHROUGH 文 [SQL Remote]」 628 ページを参照してください。

備考

UNLOAD 文では、クエリの結果セットをカンマ区切りのファイルにエクスポートできます。クエリに ORDER BY 句が指定されていなければ、結果セットは順序付けられません。

バイナリ・データ型の結果セット・カラムをアンロードする場合、UNLOAD は、*¥0xnnnn* の形式で 16 進文字列を書き込みます。 *n* は 16 進数字です。

unload-option パラメータの詳細については、「UNLOAD TABLE 文」 725 ページを参照してください。

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・ユーザをリモート・ユーザにマップしてください。外部ログインがない場合は、リモート・サーバに接続できないため再ロードが失敗します。

外部ログインの詳細については、「外部ログインの使用」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

APPEND オプションを 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 オプションを 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。このオプションのデフォルトは OFF です。

パーミッション

UNLOAD 文を実行するのに必要なパーミッションは、-gl オプションを使ってデータベース・サーバ・コマンド・ラインで設定されます。「-gl サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

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

- 「UNLOAD TABLE 文」 725 ページ
- 「OUTPUT 文 [Interactive SQL]」 623 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

UNLOAD TABLE 文

この文は、データベース・テーブルまたは実体化ビュー (Materialized View) から外部ファイルにデータをエクスポートするために使用します。

構文

UNLOAD [**FROM**] {
[**TABLE**] [*owner.*]*table-name*
| [**MATERIALIZED VIEW**] [*owner.*]*materialized-view-name* }
TO *file-name*
[*unload-option* ⋯]

unload-option :
APPEND { **ON** | **OFF** }
| **DELIMITED BY** *string*
| **ENCODING** *encoding*
| **ESCAPE CHARACTER** *character*
| **ESCAPES** { **ON** | **OFF** }
| **FORMAT** { **ASCII** | **BCP** }
| **HEXADECIMAL** { **ON** | **OFF** }
| **ORDER** { **ON** | **OFF** }
| **QUOTE** *string*
| **QUOTES** { **ON** | **OFF** }
| **ROW DELIMITED BY** *string*

file-name : { *string* | *variable* }

encoding : *string*

パラメータ

file-name　データのアンロード先となるファイル。データベース・サーバが文を実行するため、ファイル名はデータベース・サーバ・コンピュータ上のファイルを指定します。相対ファイル名は、データベース・サーバの開始ディレクトリを基準に相対パスでファイルを示します。クライアント・コンピュータへのデータのアンロードについては、「PASSTHROUGH 文 [SQL Remote]」 628 ページを参照してください。

APPEND オプション　APPEND オプションを 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 オプションを 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。このオプションのデフォルトは OFF です。

DELIMITED BY　カラム間で使用される文字列です。デフォルトのカラム・デリミタはカンマです。指定した文字列を代わりのカラム・デリミタとして使えます。ただし、文字列の最初のバイトのみ (文字) がデリミタとして使用されます。

ENCODING オプション　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、指定した文字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、データベースの文字エンコードが使用され、変換は実行されません。

SQL Anywhere でサポートされるエンコードのリストを取得する方法については、「サポートされている文字セット」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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。「on_charset_conversion_failure オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

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';
```

ESCAPES オプション　ESCAPES をオン (デフォルト) に設定すると、円記号の組み合わせを使用して、エクスポートに必要な場合の特殊文字を識別できます。

FORMAT オプション　データを ASCII 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。

HEXADECIMAL オプション　デフォルトでは、HEXADECIMAL は ON です。バイナリ・カラム値は **0x***nnnnnn*… のフォームで書き込まれます。*n* はそれぞれ 16 進数字を表します。マルチバイト文字セットを扱う場合は、HEXADECIMALON を使用することが重要です。

HEXADECIMAL オプションを使用する場合は、FORMAT ASCII オプションのみ指定してください。

ORDER オプション　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。

クラスタード・インデックスの詳細については、「クラスタード・インデックスの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

QUOTE オプション　QUOTE 句は ASCII データベースのみ用です。*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 オプション　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一重引用符で囲まれます。

ROW DELIMITED BY オプション　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文字列はカンマです。最長で 255 バイトの文字列を指定して、代替のデリミタを指定できます。たとえば、... ROW DELIMITED BY '###' ... などです。同じフォーマット要件が、他の SQL 文字列に適応されます。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。

備考

UNLOAD TABLE 文は、データベース・テーブルまたは実体化ビュー (Materialized View) 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。

UNLOAD TABLE は、テーブル全体に排他ロックを配置します。

バイナリ・データ型の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。

FORMAT と ESCAPE CHARACTER の各オプションの詳細については、「LOAD TABLE 文」 603 ページを参照してください。

パーミッション

UNLOAD 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 の場合、UNLOAD TABLE 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 の場合、DBA 権限を持っていることが必要です。
- -gl オプションが NONE の場合、UNLOAD TABLE は使用できません。

「-gl サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』を参照してください。

関連する動作

なし。

参照

- 「LOAD TABLE 文」 603 ページ
- 「OUTPUT 文 [Interactive SQL]」 623 ページ
- 「UNLOAD 文」 723 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の例は、UTF-8 エンコードのテーブル・データを mytable にアンロードします。

```
LOAD TABLE mytable TO 'mytable_data_in_utf8.dat' ENCODING 'UTF-8';
```


UPDATE 文

この文は、データベース・テーブルの既存のローを修正するために使用します。

構文 1

UPDATE [**FIRST** | **TOP** *n*] *table-list*
SET *set-item*, ⋯
[**FROM** *table-list*]
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**], ⋯]
[**OPTION(** *query-hint*, ... **)**]

構文 2

UPDATE *table-name*
SET *set-item*, ⋯
VERIFY (*column-name*, ⋯ **) VALUES (** *expression*, ⋯ **)**
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**], ⋯]
[**OPTION(** *query-hint*, ... **)**]

構文 3

UPDATE *table*
PUBLICATION *publication*
{ **SUBSCRIBE BY** *expression*
| **OLD SUBSCRIBE BY** *expression* **NEW SUBSCRIBE BY** *expression*
}
WHERE *search-condition*

set-item :
column-name [.*field-name*⋯] **=** *expression*
| **@***variable-name* **=** *expression*

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

パラメータ

UPDATE 句 テーブルは、ベース・テーブル、テンポラリ・テーブル、またはビューです。ビューを定義する SELECT 文が GROUP BY 句または集合関数を含まないか、あるいは UNION 文を伴わない場合は、ビューを更新できます。

FIRST または TOP 句 この句は主に ORDER BY 句で使用します。この句によって、WHERE 句の条件を満たすローのサブセットだけを更新できます。FIRST または TOP には入力として変数を使用できません。

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、古い値が使われます。

また、SET 句は、このフォーマットを使用する変数を割り当てるときにも使用できます。

```
SET @variable-name = expression, ...
```

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。

次に、UPDATE 文の一部の例を示します。この文は、テーブルを更新するだけでなく、変数に代入しています。

```
UPDATE T SET @var = expression1, col1 = expression2
WHERE...
```

これは次と同じです。

```
SELECT @var = expression1
FROM T
WHERE... ;
UPDATE T SET col1 = expression2
WHERE...
```

FROM 句　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。データは UPDATE キーワードのテーブル・リストの中だけで更新されます。

FROM 句を使用する場合、テーブル名を文の両方の部分において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。

この句は ansi_update_constraints が Off に設定されている場合のみ使用できます。「ansi_update_constraints オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

ジョインの詳細については、「ジョイン：複数テーブルからのデータ検索」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

詳細については、「FROM 句」 552 ページを参照してください。

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 句では序数のカラム番号を使用できません。

ansi_update_constraints オプションを Off に設定した場合を除き、ORDER BY 句に含まれるカラムは更新しないでください。「ansi_update_constraints オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

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 '*option-value*'** MATERIALIZED VIEW OPTIMIZATION 句を使用して、オプティマイザがクエリを処理するときに実体化ビュー (Materialized View) 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。「materialized_view_optimization オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **FORCE OPTIMIZATION** 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省略されます。ただし、コストベースの最適化を実行したい場合もあります。たとえば、クエリ処理時に実体化ビュー (Materialized View) 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、ビューのマッチング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、プロシージャ内の SELECT 文に FORCE OPTIMIZATION オプションを指定すると、プロシージャに対するすべての呼び出しにオプティマイザ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。

  単純なクエリとビューのマッチングに関する詳細については、「クエリ処理のフェーズ」『SQL Anywhere サーバ - SQL の使用法』と「実体化ビュー (Materialized View) によるパフォーマンスの向上」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- ***option-name* = *option-value*** 該当する文に対してのみ、パブリック・オプションやテンポラリ・オプションの設定よりも優先されるオプション設定を指定します。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 [互換性]」『SQL Anywhere サーバ - データベース管理』
  - 「max_query_tasks オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
  - 「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』

大文字と小文字の区別 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

Value で更新した CHAR 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 などと同じとみなされます。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。

ローを変更しない更新　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 トリガは、ローを対象に UPDATE が実行されるたびに起動されます。この場合、新しい値が元の値と異なるかどうかは問題ではありません。一方、AFTER UPDATE トリガは、新しい値が古い値と異なる場合にのみ起動されます。

備考

UPDATE 文の構文 1 は、1 つまたは複数のテーブルのロー内の値を修正します。構文 2 と構文 3 は SQL Remote だけに適用できます。

構文 2 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 3 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 トリガに配置され、データベース・サーバが 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 3 を含むトリガ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 3 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 3 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

カラム統計が更新されます。

参照

- 「DELETE 文」 497 ページ
- 「INSERT 文」 590 ページ
- 「FROM 句」 552 ページ
- 「ジョイン：複数テーブルからのデータ検索」 『SQL Anywhere サーバ - SQL の使用法』
- 「UPDATE (位置付け) 文 [ESQL] [SP]」 733 ページ

標準と互換性

- **SQL/2003** 構文 1 はコア機能です。ただし、FROM 句と ORDER BY 句はベンダ拡張です。構文 2 と 3 は、SQL Remote だけに対するベンダ拡張です。

  SQL/2003 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。「ansi_update_constraints オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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。

外部キーのプロパティの詳細については、「ALTER TABLE 文」 336 ページと 「CREATE TABLE 文」 460 ページを参照してください。

この例は、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 );
```

UPDATE (位置付け) 文 [ESQL] [SP]

この文は、カーソルの現在の位置でデータを修正するために使用します。

構文 1

UPDATE WHERE CURRENT OF *cursor-name*
{ **USING DESCRIPTOR** *sqlda-name* | **FROM** *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*.]*table-or-view-name* [[**AS**] *correlation-name*]

set-item :
[*correlation-name*.]*column-name* **=** *expression*
| [*owner-name*.]*table-or-view-name*.*column-name* **=** *expression*

sqlda-name : *identifier*

パラメータ

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。*式*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、変数、クエリの選択リストからの式、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。subselects、subquery 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。

備考

UPDATE 文のこのフォームは、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 ポインタは、null ポインタで、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。

関連する動作

なし。

参照

- 「DELETE 文」 497 ページ
- 「DELETE (位置付け) 文 [ESQL] [SP]」 501 ページ
- 「UPDATE 文」 728 ページ

標準と互換性

- **SQL/2003** コア機能。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拡張が含まれます。
- **Sybase** Embedded SQL の使用は Open Client/Open Server でサポートされ、プロシージャとトリガの使用は SQL Anywhere でサポートされます。

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]

この文は、データベース内のデータの修正に使用します。

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*
PUBLICATION *publication*
{ **SUBSCRIBE BY** *expression* |
  OLD SUBSCRIBE BY *expression*
  NEW SUBSCRIBE BY *expression* }
WHERE *search-condition*

expression: *value* | *subquery*

備考

構文 1 と構文 2 は SQL Remote のみに適用されます。

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。

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 文が GROUP BY 句や集合関数を含まないか、または UNION 文を伴わない場合は、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。このため、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 などと同じとみなされます。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 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 トリガに配置され、データベース・サーバが 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。

この機能の詳細については、「Contacts データベースの例における領域の再編成」 『SQL Remote』を参照してください。

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。

標準と互換性

- **SQL/2003** ベンダ拡張。

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;
```

VALIDATE 文

この文は、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テーブルや実体化ビュー (Materialized View) を検証するときに使用します。

構文 1 - テーブルと実体化ビュー (Materialized View) の検証

VALIDATE {
TABLE [*owner*.]*table-name*
| **MATERIALIZED VIEW** [*owner*.]*materialized-view-name* }
[**WITH EXPRESS CHECK**]

構文 2 - データベースの検証

VALIDATE { **CHECKSUM** | **DATABASE** }

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*

パラメータ

WITH EXPRESS CHECK　デフォルトのチェックに加え、テーブルまたは実体化ビュー (Materialized View) にあるローの数がインデックスのエントリ数と一致するかどうかをチェックします。このオプションは、各ローの各インデックスのルックアップは実行しません。また、チェックサムの検証も実行しません。小さいキャッシュを使用して大きいデータベースを検証する場合は、このオプションを使用するとパフォーマンスを大幅に改善できます。

備考

テーブルの検証には、チェックサムの検証が含まれます。また、テーブル内のロー数が、そのテーブルと関連がある各インデックスのロー数と一致することを検証します。WITH EXPRESS CHECK を指定すると、チェックサムの検証は実行されません。

VALIDATE DATABASE 文は、データベースのテーブル・ページがすべて現在のオブジェクトに所属することを検証します。また、VALIDATE DATABASE はチェックサム検証を実行しますが、インデックスは検証しません。

VALIDATE CHECKSUM 文は、データベース上で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 [サーバ・メッセージ] ウィンドウに表示されます。重要なデータベース・ページにはチェッ

クサムが含まれるため、VALIDATE CHECKSUM 文はチェックサムが無効なデータベースの場合でも有効なことがあります。

テーブルまたは実体化ビュー (Materialized View) に対して、インデックスの統計情報など、インデックスを検証するときに VALIDATE INDEX 文を使用します。VALIDATE INDEX 文によって、インデックスで参照されているすべてのローが実際に存在していることを確認します。外部キー・インデックスの場合は、対応するローがプライマリ・テーブルにあることも確認します。この検査は、VALIDATE TABLE 文によって実行される妥当性検査を補完するものです。VALIDATE INDEX 文は、指定されたインデックスについてレポートされた統計が正確であるかどうかも検証します。正確でない場合、エラーが生成されます。

警告
テーブルまたはデータベース全体の検証は、どの接続においてもデータベースを変更していない場合に実行してください。そうでない場合、実際に破損がなくても、何らかの形でデータベースが破損したことを示す重大な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 権限が必要です。

関連する動作

なし。

参照

- 「検証ユーティリティ (dbvalid)」 『SQL Anywhere サーバ - データベース管理』
- 「sa_validate システム・プロシージャ」 970 ページ
- 「データベースの妥当性の確認」 『SQL Anywhere サーバ - データベース管理』
- 「CREATE DATABASE 文」 379 ページ
- 「CREATE INDEX 文」 414 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

WAITFOR 文

この文を使用して、指定した時間の間、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

DELAY　DELAY を使用すると、処理は特定の期間だけ中断されます。

TIME　TIME を指定すると、データベース・サーバ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　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　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。ほとんどの場合、メッセージが受信されると、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。

MessageReceived プロパティの詳細については、「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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。

WAITFOR は、次の文の代わりに使用できます。

```
CALL java.lang.Thread.sleep( <time_to_wait_in_millisecs> );
```

通常は、WAITFOR TIME を使用するよりも、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。これは、スケジュールされたイベントは、専用接続で実行されるためです。

パーミッション

なし。

関連する動作

この文の実装は、待機中にワーカ・スレッドを使用します。-gn データベース・オプションで指定されたスレッドの 1 つが使用されます (デフォルトのスレッド数は 20 です)。

参照

- 「CREATE EVENT 文」 397 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

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。

```
// 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]

この文は、Embedded SQL プログラム内でのエラー処理を指定するために使用します。

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。

ソースでの位置に基づくエラー
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/2003**　コア機能。

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]

この文は、文または複合文を繰り返して実行するために使用します。

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」 614 ページ

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

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 文で WINDOW 句を使用すると、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

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。

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ができます。

この句を省略すると、SQL Anywhere は最も効率的な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。

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することができます。ウィンドウ・サイズは、現在のローの値からデータ値のオフセット範囲を指定するか (RANGE)、現在のローから行数の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、ORDER BY も使用する必要があります。この理由は、ウィンドウを作成するときに実行する計算は、値を格納する必要があるためです。さらに、ORDER BY 句には複数の式を含めることはできません。また、式の結果は日付または数値になる必要があります。

ROWS 句または RANGE 句を使用する場合、開始ローのみを使用するのであれば、現在のローがウィンドウの最終ローとして使用されます。最終ローのみを指定する場合、現在のローは最初のローとして使用されます。

- **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。

  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。

ROW 句または RANGE 句を指定しない場合、ウィンドウ・サイズは次のように決まります。

- ORDER BY 句を指定する場合、ウィンドウはパーティションの最初のロー (UNBOUNDED PRECEDING) で開始し、現在のロー (CURRENT ROW) で終了します。
- ORDER BY 句が指定されていない場合、ウィンドウはパーティションの最初のロー (UNBOUNDED PRECEDING) で開始し、パーティションの最後のロー (UNBOUNDED FOLLOWING) で終了します。

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。

目的の結果によっては、WINDOW 句のウィンドウ設定をすべて指定して、Window 関数構文内からウィンドウに名前を付ける (参照する) こともあります (たとえば、AVG() OVER *window-name*)。Window 関数の全体のウィンドウを指定して、WINDOW 句をまったく使用しないことも

できます。最終的に、Window 関数構文と WINDOW 句の定義を分割することもできます。次に例を示します。

```
AVG() OVER ( windowA
        ORDER BY expression )...
...
WINDOW windowA AS ( PARTITION BY expression )
```

この方法でウィンドウ定義を分割すると、次の制限が適用されます。

- Window 関数構文では PARTITION BY 句を使用できません。
- Window 関数構文または WINDOW 句のいずれかで ORDER BY 句を使用することはできますが、両方では使用できません。
- RANGE 句または ROWS 句を WINDOW 句に含めることはできません。

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。その他の Window 関数については、目的の操作に応じて、任意の句を使用できます。

目的の操作に応じてウィンドウを定義して使用する方法の詳細については、「ウィンドウの定義」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

参照

- 「SELECT 文」 669 ページ
- 「OLAP のサポート」『SQL Anywhere サーバ - SQL の使用法』

標準と互換性

- **SQL/2003** SQL/2003 機能 T611、T612。

例

次の例は、従業員の給料と全従業員の平均給料を State で返します。結果は State、Surname の優先順でソートされます。

```
SELECT EmployeeID, Surname, Salary, State,
  AVG( Salary ) OVER SalaryWindow
FROM Employees
WINDOW SalaryWindow AS ( PARTITION BY State )
ORDER BY State, Surname;
```


WRITETEXT 文 [T-SQL]

既存の text または image カラムの、ログなしの対話型更新を許可します。

構文

WRITETEXT *table-name.column-name*
text-pointer [**WITH LOG**] *data*

備考

既存のテキストまたはイメージ・カラムを更新します。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]」 639 ページ
- 「TEXTPTR 関数 [テキストとイメージ]」 267 ページ

標準と互換性

- **SQL/2003** Transact-SQL 拡張。

例

次のコード・フラグメントは、WRITETEXT 文の使い方を示します。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 column_name カラムの内容を値 newdata で置換します。

```
EXEC SQL create variable textpointer binary(16);
EXEC SQL set textpointer =
  (  SELECT textptr(column_name)
     FROM table_name WHERE ID = 5 );
EXEC SQL writetext table_name.column_name
  textpointer 'newdata';
```

パート II. システム・オブジェクト

この項では、SQL Anywhere のテーブル、ビュー、プロシージャについて説明します。

第 5 章

テーブル

目次

システム・テーブル

すべてのデータベースの構造は、多数のシステム・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 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 コマンドは使用できません。また、ALTER TABLE と DROP コマンドを使って、これらのテーブルの構造を変更することもできません。

SQL Anywhere のシステム・テーブルは、対応するビュー経由で公開されます。

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 ID と今日の日付を取り出す例を示します。

```
SELECT USER, today(*) FROM SYS.DUMMY;
```

FROM 句での SYS.DUMMY はオプションです。FROM 句でテーブルが指定されない場合は、テーブルは SYS.DUMMY テーブルとみなされます。上記の例は、次のように記述できます。

```
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、同様のユーザ作成テーブルからの読み取りコストよりも低いコストです。これは、SYS.DUMMY のテーブル・ページには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・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。「ロー・コンストラクタ・アルゴリズム」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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の記事に関する記述です。「SYSARTICLE システム・ビュー」 782 ページを参照してください。

ISYSARTICLECOL システム・テーブル

ISYSARTICLECOL システム・テーブルの各ローは記事のカラムを識別します。「SYSARTICLECOL システム・ビュー」 783 ページを参照してください。

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・サーバの機能を識別します。「SYSCAPABILITY システム・ビュー」 783 ページを参照してください。

ISYSCAPABILITYNAME システム・テーブル

ISYSCAPABILITYNAME システム・テーブルの各ローは、ISYSCAPABILITY システム・テーブルで定義されている機能に名前を付けます。「SYSCAPABILITYNAME システム・ビュー」 784 ページを参照してください。

ISYSCHECK システム・テーブル

ISYSCHECK システム・テーブルの各ローは、テーブルの名前付き検査制約を識別します。「SYSCHECK システム・ビュー」 785 ページを参照してください。

ISYSCOLPERM システム・テーブル

ISYSCOLPERM システム・テーブルの各ローには、カラムの UPDATE、SELECT、または REFERENCES パーミッションを記述します。「SYSCOLPERM システム・ビュー」 785 ページを参照してください。

ISYSCOLSTAT システム・テーブル

ISYSCOLSTAT システム・テーブルには、オプティマイザが使用するカラムの統計情報が含まれます。「SYSCOLSTAT システム・ビュー」 786 ページを参照してください。

ISYSCONSTRAINT システム・テーブル

ISYSCONSTRAINT システム・テーブルの各ローには、システム・テーブル以外のすべてのテーブルの名前付き制約を記述します。「SYSCONSTRAINT システム・ビュー」 787 ページを参照してください。

ISYSDEPENDENCY システム・テーブル

ISYSDEPENDENCY システム・テーブルの各ローには、テーブルとビューの依存性を記述します。「SYSDEPENDENCY システム・ビュー」 788 ページを参照してください。

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**」ともいいます) 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。「SYSDOMAIN システム・ビュー」 789 ページを参照してください。

ISYSEVENT システム・テーブル

ISYSEVENT システム・テーブルの各ローには、CREATE EVENT で作成されたイベントを記述します。「SYSEVENT システム・ビュー」 789 ページを参照してください。

ISYSEVENTTYPE システム・テーブル

ISYSEVENTTYPE システム・テーブルは、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。「SYSEVENTTYPE システム・ビュー」 791 ページを参照してください。

ISYSEXTERNLOGIN システム・テーブル

ISYSEXTERNLOGIN システム・テーブルの各ローには、リモート・データ・アクセスの外部ログインを記述します。「SYSEXTERNLOGIN システム・ビュー」 791 ページを参照してください。

ISYSFILE システム・テーブル

ISYSFILE システム・テーブルの各ローには、データベースの DB 領域を記述します。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。「SYSFILE システム・ビュー」 792 ページを参照してください。

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に関する記述です。「SYSFKEY システム・ビュー」 793 ページを参照してください。

ISYSGROUP システム・テーブル

ISYSGROUP システム・テーブルの各ローではグループのメンバを定義します。このテーブルは、グループとメンバの多対多の関係を示します。「SYSGROUP システム・ビュー」 794 ページを参照してください。

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るタイミングを示します。「SYSHISTORY システム・ビュー」 795 ページを参照してください。

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 1 つのインデックスに関する記述です。「SYSIDX システム・ビュー」 796 ページを参照してください。

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に関する記述です。「SYSINDEXES システム・ビュー」 798 ページを参照してください。

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 ファイルを定義しています。「SYSJAR システム・ビュー」 799 ページを参照してください。

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ています。「SYSJARCOMPONENT システム・ビュー」 800 ページを参照してください。

ISYSJAVACLASS システム・テーブル

ISYSJAVACLASS システム・テーブルの各ローには Java クラスが記述されています。「SYSJAVACLASS システム・ビュー」 800 ページを参照してください。

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・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 権限を持つユーザだけが表示できます。「SYSLOGINMAP システム・ビュー」 801 ページを参照してください。

ISYSMVOPTION システム・テーブル

ISYSMVOPTION システム・テーブルの各ローには、実体化ビュー (Materialized View) のオプションを記述しています。「SYSMVOPTION システム・ビュー」 802 ページを参照してください。

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実体化ビュー (Materialized View) の名前を指定します。「SYSMVOPTIONNAME システム・ビュー」 803 ページを参照してください。

ISYSOBJECT システム・テーブル

ISYSOBJECT システム・ビューの各ローにはオブジェクトを記述します。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、プロシージャなどがあります。「SYSOBJECT システム・ビュー」 803 ページを参照してください。

ISYSOPTION システム・テーブル

ISYSOPTION システム・テーブルの各ローには、1 人のユーザ ID のオプション設定を記述します。オプション設定は SET コマンドで ISYSOPTION テーブルに保存され、各ユーザはオプションごとに独自の設定を指定します。「SYSOPTION システム・ビュー」 805 ページを参照してください。

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。「SYSOPTSTAT システム・ビュー」 805 ページを参照してください。

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 1 つの物理インデックスに関する記述です。「SYSPHYSIDX システム・ビュー」 806 ページを参照してください。

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に関する記述です。「SYSPROCEDURE システム・ビュー」 807 ページを参照してください。

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に対するパラメータの記述です。「SYSPROCPARM システム・ビュー」 809 ページを参照してください。

ISYSPROCPERM システム・テーブル

ISYSPROCPERM テーブルの各ローは、1 つのプロシージャを呼び出すための、ユーザに付与された 1 つのパーミッションに関する記述です。「SYSPROCPERM システム・ビュー」 810 ページを参照してください。

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 テーブルに関する記述です。「SYSPROXYTAB システム・ビュー」 811 ページを参照してください。

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 パブリケーションに関する記述です。「SYSPUBLICATION システム・ビュー」 811 ページを参照してください。

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) に関する記述です。「SYSREMARK システム・ビュー」 812 ページを参照してください。

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・パラメータの値に関する記述です。「SYSREMOTEOPTION システム・ビュー」 813 ページを参照してください。

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・パラメータのいずれかに関する記述です。「SYSREMOTEOPTIONTYPE システム・ビュー」 814 ページを参照してください。

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。「SYSREMOTETYPE システム・ビュー」 814 ページを参照してください。

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 ID (サブスクライバ) を記述します。そのユーザに送信された SQL Remote メッセージと、そのユーザ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。「SYSREMOTEUSER システム・ビュー」 815 ページを参照してください。

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。「SYSSCHEDULE システム・ビュー」 817 ページを参照してください。

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・サーバに関する記述です。「SYSSERVER システム・ビュー」 818 ページを参照してください。

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。「SYSSOURCE システム・ビュー」 819 ページを参照してください。

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。「SYSSQLSERVERTYPE システム・ビュー」 819 ページを参照してください。

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 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関する記述です。「SYSSUBSCRIPTION システム・ビュー」 820 ページを参照してください。

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。「SYSSYNC システム・ビュー」 821 ページを参照してください。

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。「SYSSYNCSCRIPT システム・ビュー」 822 ページを参照してください。

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 1 つのテーブルに関する記述です。「SYSTAB システム・ビュー」 823 ページを参照してください。

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に関する記述です。「SYSTABCOL システム・ビュー」 826 ページを参照してください。

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 ID (「**grantor**」)、そしてパーミッションを与えられるユーザ ID (「**grantee**」) に対応します。「SYSTABLEPERM システム・ビュー」 828 ページを参照してください。

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に関する記述です。「SYSTRIGGER システム・ビュー」 830 ページを参照してください。

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。「SYSTYPEMAP システム・ビュー」 832 ページを参照してください。

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に関する記述です。「SYSUSER システム・ビュー」 832 ページを参照してください。

ISYSUSERAUTHORITY システム・テーブル

ISYSUSERAUTHORITY システム・テーブルの各ローは、ユーザに付与された権限に関する記述です。「SYSUSERAUTHORITY システム・ビュー」 833 ページを参照してください。

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定義メッセージが入っています。「SYSUSERMESSAGE システム・ビュー」 834 ページを参照してください。

ISYSUSERTYPE システム・テーブル

ISYSUSERTYPE システム・テーブルの各ローは、ユーザ定義のデータ型に関する記述です。「SYSUSERTYPE システム・ビュー」 834 ページを参照してください。

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 1 つのビューに関する記述です。「SYSVIEW システム・ビュー」 835 ページを参照してください。

ISYSWEBSERVICE システム・テーブル

ISYSWEBSERVICE システム・テーブルの各ローは、Web サービスに関する記述です。「SYSWEBSERVICE システム・ビュー」 837 ページを参照してください。

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 - SQL の使用法』
- 「診断トレーシングを使用した詳細なアプリケーション・プロファイリング」『SQL Anywhere サーバ - SQL の使用法』

sa_diagnostic_auxiliary_catalog テーブル

sa_diagnostic_auxiliary_catalog テーブルは、dbo ユーザが所有し、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、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	カラム制約	テーブル制約
original_object_id	UNSIGNED BIGINT	NOT NULL	プライマリ・キー
local_object_id	UNSIGNED BIGINT	NOT NULL	ユニーク
pages_if_table	UNSIGNED INT		
rows_if_table	UNSIGNED BIGINT		

original_object_id　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　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　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　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]」 271 ページ
- 「インデックス・コンサルタント」 『SQL Anywhere サーバ - SQL の使用法』

sa_diagnostic_blocking テーブル

sa_diagnostic_blocking テーブルは、dbo ユーザが所有し、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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	外部キーは sa_diagnostic_cursor を参照します。 外部キーは sa_diagnostic_request を参照します。
lock_id	UNSIGNED BIGINT	NOT NULL	
request_id	UNSIGNED BIGINT		外部キーは sa_diagnostic_request を参照します。
cursor_id	UNSIGNED BIGINT		外部キーは sa_diagnostic_cursor を参照します。
original_table_object_id	UNSIGNED BIGINT		
rowid	UNSIGNED BIGINT		
block_time	TIMESTAMP	NOT NULL	
unblock_time	TIMESTAMP		
blocked_by	UNSIGNED INT	NOT NULL	

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

lock_id　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てロックの ID。それ以外の場合は NULL。

request_id　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　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　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　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　ブロックが発生した時間。

unblock_time　ブロックが終了した時間。

blocked_by　ロックを保持し、ブロックを引き起こした接続の ID。

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 - SQL の使用法』
- 「ロックの仕組み」『SQL Anywhere サーバ - SQL の使用法』

sa_diagnostic_cachecontents テーブル

sa_diagnostic_cachecontents テーブルは、dbo ユーザが所有します。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に、各テーブルのロー数だけでなく、各テーブルのテーブル・ページ数をキャッシュに記録します。オプティマイザはこの情報を使用して、クエリを元々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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	
"time"	TIMESTAMP	NOT NULL	プライマリ・キー
original_table_object_id	UNSIGNED BIGINT	NOT NULL	プライマリ・キー

カラム名	カラム型	カラム制約	テーブル制約
pages_in_cache	UNSIGNED INT	NOT NULL	
num_table_pages	UNSIGNED INT	NOT NULL	
num_table_rows	UNSIGNED BIGINT	NOT NULL	

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

"time"　キャッシュのスナップショットを撮った時間。

original_table_object_id　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　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　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　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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	プライマリ・キー
connection_number	UNSIGNED INT		プライマリ・キー
connection_name	LONG VARCHAR		
user_name	LONG VARCHAR		
comm_link	CHAR(40)		
node_address	LONG VARCHAR		
appinfo	LONG VARCHAR		

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

connection_number　データベースへのユーザの接続を識別するためにデータベース・サーバが割り当てる番号。この値は、Number 接続プロパティの値を反映しています。

connection_name　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　データベースに接続するユーザ名。

comm_link　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　クライアント／サーバ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　クライアント・コンピュータの 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。

参照

- 「接続レベルのプロパティ」『SQL Anywhere サーバ - 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記述します。

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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは sa_diagnostic_query を参照します。
cursor_id	UNSIGNED BIGINT	NOT NULL	プライマリ・キー
query_id	UNSIGNED BIGINT	NOT NULL	外部キーは sa_diagnostic_query を参照します。
isolation_level	TINYINT		
flags	UNSIGNED INT		

カラム名	カラム型	カラム制約	テーブル制約
forward_fetches	UNSIGNED INT		
reverse_fetches	UNSIGNED INT		
absolute_fetches	UNSIGNED INT		
first_fetch_time_ms	UNSIGNED INT		
total_fetch_time_ms	UNSIGNED INT		
plan_xml	LONG VARCHAR		

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

cursor_id　カーソルを識別するユニークな番号。

query_id　このカーソルの範囲にあるクエリを識別します。

isolation_level　このカーソルを開いた独立性レベル。

flags　内部的に使用。

forward_fetches　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　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　カーソルで実行された絶対フェッチの数。

first_fetch_time_ms　最初のローをフェッチするときにかかる時間。

total_fetch_time_ms　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　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

- 「カーソルの概要」『SQL Anywhere サーバ - 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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	
snapshot_id	UNSIGNED BIGINT	NOT NULL	
snapshot_at	TIMESTAMP	NOT NULL	
waiter	UNSIGNED INT	NOT NULL	
request_id	UNSIGNED BIGINT		
original_table_object_id	UNSIGNED BIGINT		
rowid	UNSIGNED BIGINT		
owner	UNSIGNED INT	NOT NULL	
rollback_operation_count	UNSIGNED INT	NOT NULL	

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

snapshot_id　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　デッドロックが発生した時刻。

waiter　このローが表す接続の接続数。

request_id　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　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　この接続がブロックされたときのローのレコード ID。

owner　このローをロックした接続の接続数。

rollback_operation_count　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 - SQL の使用法』

sa_diagnostic_hostvariable テーブル

sa_diagnostic_hostvariable テーブルは dbo ユーザが所有しています。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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは sa_diagnostic_request を参照します。
request_id	UNSIGNED BIGINT	NOT NULL	プライマリ・キー 外部キーは sa_diagnostic_request を参照します。
cursor_id	UNSIGNED BIGINT		
hostvar_num	UNSIGNED SMALLINT	NOT NULL	プライマリ・キー
hostvar_type	UNSIGNED TINYINT	NOT NULL	
hostvar_value	LONG VARCHAR		

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

request_id　ホスト変数が所属する要求の ID。

cursor_id　ホスト変数が保持するカーソルの ID。

hostvar_num　SQL 文のホスト変数の序数位置。

hostvar_type　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　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

- 「ホスト変数の使用」『SQL Anywhere サーバ - 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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	
request_id	UNSIGNED BIGINT		
rowvariable_id	UNSIGNED INT		
variable_domain	UNSIGNED SMALLINT		
variable_name	CHAR(128)		
variable_value	LONG VARCHAR		

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

request_id　内部変数を含む要求 ID。

rowvariable_id　この値のロー変数のカラム数。

variable_domain　内部変数のデータ型。

variable_name　内部変数の名前。

variable_value　内部変数の値を表す文字列。

参照

- 「ローカル変数」 36 ページ

sa_diagnostic_query テーブル

sa_diagnostic_query テーブルは dbo ユーザが所有しています。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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	プライマリ・キー。外部キーは sa_diagnostic_statement を参照します。

カラム名	カラム型	カラム制約	テーブル制約
query_id	UNSIGNED BIGINT	NOT NULL	プライマリ・キー。外部キーは sa_diagnostic_statement を参照します。
statement_id	UNSIGNED BIGINT	NOT NULL	
user_object_id	UNSIGNED BIGINT	NOT NULL	
start_time	TIMESTAMP	NOT NULL	
cache_size_bytes	UNSIGNED BIGINT		
optimization_goal	TINYINT		
optimization_level	TINYINT		
user_estimates	TINYINT		
optimization_workload	TINYINT		
available_requests	TINYINT		
active_requests	TINYINT		
max_tasks	TINYINT		
used_bypass	TINYINT		
estimated_cost_ms	TINYINT		
plan_explain	LONG VARCHAR		
plan_xml	LONG VARCHAR		
sql_rewritten	LONG VARCHAR		

logging_session_id　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　クエリをユニークに識別する番号。

statement_id　クエリで文をユニークに識別する番号。

user_object_id　このクエリが実行されたときのユーザのオブジェクト ID。クエリをプロシージャから実行した場合、これはプロシージャ所有者の ユーザ ID になります。

start_time　クエリを最適化した時刻。

cache_size_bytes　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　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。

このカラムに使用できる値については、「optimization_goal オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

optimization_level SQL Anywhere クエリ・オプティマイザが 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。

このカラムに使用できる値については、「optimization_level オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

user_estimates クエリの述部に含まれるユーザ選択性推定をクエリ・オプティマイザ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。

このカラムに使用できる値については、「user_estimates オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

optimization_workload 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。

このカラムに使用できる値については、「optimization_workload オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

available_requests クエリ内並列処理のレベルを内部的に計算するときに使用します。

active_requests クエリ内並列処理のレベルを内部的に計算するときに使用します。

max_tasks クエリ内並列処理のレベルを内部的に計算するときに使用します。

used_bypass 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 推定コスト (ミリ秒)。

plan_explain このクエリのテキスト・プラン表記。

plan_xml クエリのグラフィカル・プラン表記 (記録された場合)。

sql_rewritten 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 - データベース管理』
- 「オプティマイザの仕組み」『SQL Anywhere サーバ - SQL の使用法』

sa_diagnostic_request テーブル

sa_diagnostic_request テーブルは、dbo ユーザが所有します。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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは sa_diagnostic_connection を参照します。 外部キーは sa_diagnostic_cursor を参照します。 外部キーは sa_diagnostic_query を参照します。 外部キーは sa_diagnostic_statement を参照します。
request_id	UNSIGNED BIGINT	NOT NULL	プライマリ・キー
start_time	TIMESTAMP	NOT NULL	
finish_time	TIMESTAMP	NOT NULL	
duration_ms	UNSIGNED INT	NOT NULL	
connection_number	UNSIGNED INT		外部キーは sa_diagnostic_connection を参照します。
request_type	UNSIGNED SMALLINT		
statement_id	UNSIGNED BIGINT		外部キーは sa_diagnostic_statement を参照します。
query_id	UNSIGNED BIGINT		外部キーは sa_diagnostic_query を参照します。
cursor_id	UNSIGNED BIGINT		外部キーは sa_diagnostic_cursor を参照します。

カラム名	カラム型	カラム制約	テーブル制約
sql_code	SMALLINT		

logging_session_id　要求が発生したときのロギング・セッション。

request_id　要求をユニークに識別する番号。

start_time　イベントが開始した時間。

finish_time　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　イベントの継続時間 (ミリ秒)。

connection_number　イベントをトリガさせた接続の ID。

request_type　要求の型。次のような値があります。

値	説明
1	新規トレーシング・セッションの開始
2	文の実行
3	カーソルが開く
4	カーソルが閉じる
5	接続
6	切断

statement_id　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　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　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　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。

sa_diagnostic_statement テーブル

sa_diagnostic_statement テーブルは dbo ユーザが所有します。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、トリガ、ユーザ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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	プライマリ・キー
statement_id	UNSIGNED BIGINT	NOT NULL	プライマリ・キー
database_object	UNSIGNED BIGINT		
line_number	UNSIGNED SMALLINT		
signature	UNSIGNED INT		
statement_text	LONG VARCHAR	NOT NULL	

logging_session_id　文が送信されたときのロギング・セッション。

statement_id　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　文がプロシージャ、トリガ、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　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序数位置を反映します。

signature　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　文のテキスト。

sa_diagnostic_statistics テーブル

sa_diagnostic_statistics テーブルは、dbo ユーザが所有します。また、サーバで維持されているパフォーマンス・カウンタ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	カラム制約	テーブル制約
logging_session_id	UNSIGNED INT	NOT NULL	
"time"	TIMESTAMP	NOT NULL	

カラム名	カラム型	カラム制約	テーブル制約
counter_id	UNSIGNED SMALLINT	NOT NULL	
type	TINYINT	NOT NULL	
connection_number	UNSIGNED INT	NOT NULL	
counter_value	UNSIGNED INT	NOT NULL	

logging_session_id　診断情報を集めているときに、一意にロギング・セッションを識別する番号。

"time"　パフォーマンス・カウンタ値をキャプチャした時刻。

counter_id　パフォーマンス・カウンタ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　データベース、サーバ、または接続の統計情報のいずれかを示します。サーバの場合は 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　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　パフォーマンス・カウンタの値。

参照

- 「PROPERTY_NAME 関数 [システム]」 219 ページ

sa_diagnostic_tracing_level テーブル

sa_diagnostic_tracing_level テーブルは、dbo ユーザが所有します。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 LEVELS 文を使用して作成されます。

カラム

カラム名	カラム型	カラム制約	テーブル制約
id	UNSIGNED INT	NOT NULL	プライマリ・キー
scope	CHAR(32)	NOT NULL	
識別子	CHAR(128)		

カラム名	カラム型	カラム制約	テーブル制約
trace_type	CHAR(32)	NOT NULL	
trace_condition	CHAR(32)		
value	UNSIGNED INT		
enabled	BIT	NOT NULL	

scope　以下に示す診断トレーシングの範囲。各スコープの詳細については、「診断トレーシングのスコープ」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- DATABASE
- ORIGIN
- USER
- CONNECTION_NAME
- CONNECTION_NUMBER
- FUNCTION
- PROCEDURE
- EVENT
- TRIGGER
- TABLE

id　内部でのみ使用されます。

識別子　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。

trace_type　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。各トレース型の詳細については、「診断トレーシングのタイプ」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- VOLATILE_STATISTICS
- NONVOLATILE_STATISTICS
- CONNECTION_STATISTICS
- BLOCKING
- PLANS
- PLANS_WITH_STATISTICS

- STATEMENTS
- STATEMENTS_WITH_VARIABLES
- OPTIMIZATION_LOGGING
- OPTIMIZATION_LOGGING_WITH_PLANS

condition　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。各条件の詳細については、「診断トレーシング条件」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- NONE または NULL
- SAMPLE_EVERY
- ABSOLUTE_COST
- RELATIVE_COST_DIFFERENCE

condition_value　*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* は予測実行コストと実際の実行コストの差異 (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　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

- 「ATTACH TRACING 文」 348 ページ
- 「REFRESH TRACING LEVEL 文」 642 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。

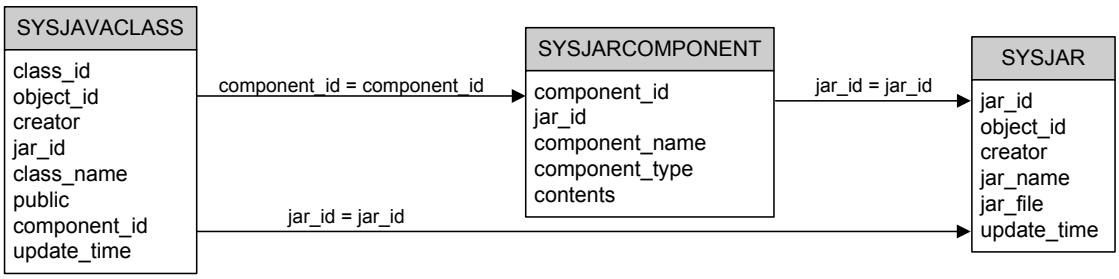
sa_rowgenerator システム・プロシージャでも一定範囲の数値を生成できます。sa_rowgenerator システム・プロシージャの使用方法と例については、「sa_rowgenerator システム・プロシージャ」 944 ページを参照してください。

カラム名	カラム型	カラム制約	基礎となるテーブルの制約
row_num	SMALLINT	NOT NULL	

row_num　1 ～ 255 の値

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・システム・テーブル」 『Mobile Link - サーバ管理』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。

第 6 章

ビュー

目次

Sybase Central のシステム・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。場合によっては、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・テーブルのカラムを含めることもできます。

将来的な SQL Anywhere カタログとの互換性を保つために、アプリケーションからはシステム・ビューを使用すること、変化する可能性がある基礎となるシステム・テーブルは使用しないことが必要です。

ビュー定義を含むシステム・ビューについての詳細は、Sybase Central を参照してください。

- システム・ビューを表示するには、接続しているデータベースを右クリックし、[所有者別にオブジェクトをフィルタ] を選択してから [SYS] を選択します。データベースの [ビュー] フォルダを開きます。
- ビュー定義を表示するには、左ウィンドウ枠でビューを選択してから、右ウィンドウ枠で [SQL] タブをクリックします。
- データを表示するには、左側のウィンドウ枠にある View フォルダを開き、ビューを選択します。右側のウィンドウ枠で、[データ] タブをクリックします。

SYSARTICLE システム・ビュー

SYSARTICLE システムの各ローは、パブリケーションのアーティクルを記述します。このビューの基礎となるシステム・テーブルは ISYSARTICLE です。

カラム

カラム名	カラム型	カラム制約
publication_id	UNSIGNED INT	NOT NULL
table_id	UNSIGNED INT	NOT NULL
where_expr	LONG VARCHAR	
subscribe_by_expr	LONG VARCHAR	
query	CHAR(1)	NOT NULL
alias	VARCHAR(256)	

publication_id　このアーティクルが含まれるパブリケーション。

table_id　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　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　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　データベース・サーバにアーティクル・タイプ情報を示します。

alias　アーティクルのエイリアス。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (publication_id, table_id)

FOREIGN KEY (publication_id) は SYS.ISYSPUBLICATION (publication_id) を参照

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

SYSARTICLECOL システム・ビュー

SYSARTICLECOL システム・ビューの各ローは記事のカラムを識別します。このビューの基礎となるシステム・テーブルは ISYSARTICLE です。

カラム

カラム名	カラム型	カラム制約
publication_id	UNSIGNED INT	NOT NULL
table_id	UNSIGNED INT	NOT NULL
column_id	UNSIGNED INT	NOT NULL

publication_id　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　カラムが属するテーブル。

column_id　SYSTABCOL システム・ビューのカラム識別子。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (publication_id, table_id, column_id)

FOREIGN KEY (publication_id) は SYS.ISYSPUBLICATION (publication_id) を参照

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

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・サーバの機能を識別します。このビューの基礎となるシステム・テーブルは ISYSCAPABILITY です。

カラム

カラム名	カラム型	カラム制約
capid	INTEGER	NOT NULL
srvid	UNSIGNED INT	NOT NULL
capvalue	CHAR(128)	NOT NULL

capid　SYSCAPABILITYNAME システム・ビューに表示されている機能 ID。

srvid　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。

capvalue　機能の値。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (capid, srvid)

FOREIGN KEY (srvid) は SYS.ISYSSERVER (srvid) を参照

FOREIGN KEY (capid) は SYS.ISYSCAPABILITYNAME (capid) を参照

参照

- 「SYSCAPABILITYNAME システム・ビュー」 784 ページ

SYSCAPABILITYNAME システム・ビュー

SYSCAPABILITYNAME システム・ビューの各ローは、SYSCAPABILITY システム・ビューで定義されている機能に名前を付けます。このビューの基礎となるシステム・テーブルは ISYSCAPABILITYNAME です。

カラム

カラム名	カラム型	カラム制約
capid	INTEGER	NOT NULL
capname	CHAR(128)	NOT NULL

capid　機能をユニークに識別する番号。

capname　機能の名前。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (capid)

参照

- 「SYSCAPABILITY システム・ビュー」 783 ページ

SYSCHECK システム・ビュー

SYSCHECK システム・ビューの各ローには、テーブルで名前付きのチェック制約の定義が記述されます。このビューの基礎となるシステム・テーブルは ISYSCHECK です。

カラム

カラム名	カラム型	カラム制約
check_id	UNSIGNED INT	NOT NULL
check_defn	LONG VARCHAR	NOT NULL

check_id　データベースの制約をユニークに識別する番号。

check_defn　CHECK 式。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (check_id)

FOREIGN KEY (check_id) は SYS.ISYSCONSTRAINT (constraint_id) を参照

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。このビューの基礎となるシステム・テーブルは ISYSCOLPERM です。

カラム

カラム名	カラム型	カラム制約
table_id	UNSIGNED INT	NOT NULL
grantee	UNSIGNED INT	NOT NULL
grantor	UNSIGNED INT	NOT NULL
column_id	UNSIGNED INT	NOT NULL
privilege_type	SMALLINT	NOT NULL
is_grantable	CHAR(1)	NOT NULL

table_id　カラムが属するテーブルのテーブル番号。

grantee　カラムにパーミッションを与えられたユーザ ID のユーザ番号。パーミッションの grantee が特別な PUBLIC ユーザ ID のユーザ番号の場合、パーミッションはすべてのユーザ ID に与えられます。

grantor　パーミッションを付与するユーザ ID のユーザ番号。

column_id　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　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 (Y/N)　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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_id, grantee, grantor, column_id, privilege_type)

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

FOREIGN KEY (grantor) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (grantee) は SYS.ISYSUSER (user_id) を参照

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。このビューの基礎となるシステム・テーブルは ISYSCOLSTAT です。

注意
データベースを暗号化する場合、またはテーブルの暗号化が有効な場合、基礎となるシステム・テーブル ISYSCOLSTAT は暗号化されます。基礎となるデータを公開する可能性があるヒストグラム情報が含まれるためです。

カラム

カラム名	カラム型	カラム制約
table_id	UNSIGNED INT	NOT NULL
column_id	UNSIGNED INT	NOT NULL
format_id	SMALLINT	NOT NULL
update_time	TIMESTAMP	NOT NULL
density	FLOAT	NOT NULL
max_steps	SMALLINT	NOT NULL
actual_steps	SMALLINT	NOT NULL

カラム名	カラム型	カラム制約
step_values	LONG BINARY	
frequencies	LONG BINARY	

table_id　このカラムが属するテーブルまたは実体化ビュー (Materialized View) をユニークに識別する番号。

column_id　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　システムでのみ使用。

update_time　このカラムの統計情報が最後に更新された時刻。

density　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　システムでのみ使用。

actual_steps　システムでのみ使用。

step_values　システムでのみ使用。

frequencies　システムでのみ使用。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_id, column_id)

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

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 ISYSCONSTRAINT です。

カラム

カラム名	カラム型	カラム制約
constraint_id	UNSIGNED INT	NOT NULL
constraint_type	CHAR(1)	NOT NULL
ref_object_id	UNSIGNED BIGINT	NOT NULL
table_object_id	UNSIGNED BIGINT	NOT NULL
constraint_name	CHAR(128)	NOT NULL

constraint_id　制約のユニークな ID。

constraint_type　制約の型。

- C - カラム・チェックの制約。
- T - テーブルの制約。
- P - プライマリ・キー。
- F - 外部キー。
- U - 一意性制約。

ref_object_id　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　制約の適用対象となるテーブルのテーブル ID。

constraint_name　制約名。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (constraint_id)

FOREIGN KEY (ref_object_id) は SYS.ISYSOBJECT (object_id) を参照

FOREIGN KEY (table_object_id) は SYS.ISYSOBJECT (object_id) を参照

UNIQUE (table_object_id, constraint_name)

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 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・サーバが、テーブル、ビュー、実体化ビュー (Materialized View)、カラムの依存性を追跡します。

カラム

カラム名	カラム型	カラム制約
ref_object_id	UNSIGNED BIGINT	NOT NULL
dep_object_id	UNSIGNED BIGINT	NOT NULL

ref_object_id　参照オブジェクトのオブジェクト ID。

dep_object_id　参照オブジェクトの ID。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (ref_object_id, dep_object_id)

FOREIGN KEY (ref_object_id) は SYS.ISYSOBJECT (object_id) を参照

FOREIGN KEY (dep_object_id) は SYS.ISYSOBJECT (object_id) を参照

参照

- 「sa_dependent_views システム・プロシージャ」 890 ページ
- 「ビューの依存性」 『SQL Anywhere サーバ - 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。このビューの基礎となるシステム・テーブルは ISYSDOMAIN です。

カラム

カラム名	カラム型	カラム制約
domain_id	SMALLINT	NOT NULL
domain_name	CHAR(128)	NOT NULL
type_id	SMALLINT	NOT NULL
"precision"	SMALLINT	

domain_id　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　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)。

type_id　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"　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。

基礎となるシステム・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記述します。このビューの基礎となるシステム・テーブルは ISYSEVENT です。

カラム

カラム名	カラム型	カラム制約
event_id	UNSIGNED INT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
creator	UNSIGNED INT	NOT NULL
event_name	VARCHAR(128)	NOT NULL
enabled	CHAR(1)	NOT NULL
location	CHAR(1)	NOT NULL
event_type_id	UNSIGNED INT	
action	LONG VARCHAR	
external_action	LONG VARCHAR	
condition	LONG VARCHAR	
remarks	LONG VARCHAR	
source	LONG VARCHAR	

event_id　各イベントに割り当てられたユニークな番号。

object_id　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　イベントの所有者のユーザ番号。ユーザ名は SYSUSER システム・ビューで確認できます。

event_name　イベント名。

enabled (Y/N)　イベントが起動できるかどうかを示します。

location　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

event_type_id　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　イベント・ハンドラ定義。

external_action　システムでのみ使用。

condition　イベント・ハンドラの起動を制御するのに使用される条件。

remarks　イベントの注釈。このカラムは ISYSREMARK に由来します。

source　イベントの元のソース。このカラムは ISYSSOURCE に由来します。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (event_id)

FOREIGN KEY (event_type_id) は SYS.ISYSEVENTTYPE (event_type_id) を参照

FOREIGN KEY (creator) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

UNIQUE (index_name, table_id, index_category)

参照

- 「SYSEVENTTYPE システム・ビュー」 791 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。このビューの基礎となるシステム・テーブルは ISYSEVENTTYPE です。

カラム

カラム名	カラム型	カラム制約
event_type_id	UNSIGNED INT	NOT NULL
name	VARCHAR(128)	NOT NULL
description	LONG VARCHAR	

event_type_id　各イベント型に割り当てられたユニークな番号。

name　システム・イベント型の名前。

description　システム・イベント型の説明。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (event_type_id)

UNIQUE (name)

参照

- 「SYSEVENT システム・ビュー」 789 ページ

SYSEXTERNLOGIN システム・ビュー

SYSEXTERNLOGIN システム・ビューの各ローには、リモート・データ・アクセスの外部ログインを記述します。このビューの基礎となるシステム・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、このビューの基礎となるテーブルになります。

カラム

カラム名	カラム型	カラム制約
user_id	UNSIGNED INT	NOT NULL
srvid	UNSIGNED INT	NOT NULL
remote_login	VARCHAR(128)	
remote_password	VARBINARY(128)	

user_id　ローカル・データベースでのユーザ ID。

srvid　SYSSERVER システム・ビューにリストされているリモート・サーバ。

remote_login　このユーザのリモート・サーバのログイン名。

remote_password　このユーザのリモート・サーバのパスワード。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (user_id, srvid)

FOREIGN KEY (user_id) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (srvid) は SYS.ISYSSERVER (srvid) を参照

SYSFILE システム・ビュー

SYSFILE システム・ビューの各ローには、データベースの DB 領域を記述します。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。このビューの基礎となるシステム・テーブルは ISYSFILE です。

SQL Anywhere は、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・ファイルの DB 領域を自動的に作成します。ただし、テンポラリ・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。「事前定義の DB 領域」『SQL Anywhere サーバ - データベース管理』を参照してください。

カラム

カラム名	カラム型	カラム制約
file_id	SMALLINT	NOT NULL
file_name	LONG VARCHAR	NOT NULL

カラム名	カラム型	カラム制約
dbspace_name	CHAR(128)	NOT NULL
store_type	INTEGER	

file_id　データベースの各ファイルには、ユニークな番号が割り当てられています。SYSTEM 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 file_id があります。

file_name　DB 領域のファイル名。SYSTEM の DB 領域の場合、データベースを作成したときに、値はデータベース・ファイルの名前です。また、この値は参照情報であり、変更できません。その他の 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-file-name*'

dbspace_name　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　このフィールドは内部で使用されます。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (file_id)

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の制約に関する記述です。このビューの基礎となるシステム・テーブルは ISYSFKEY です。

カラム

カラム名	カラム型	カラム制約
foreign_table_id	UNSIGNED INT	NOT NULL
foreign_index_id	UNSIGNED INT	NOT NULL
primary_table_id	UNSIGNED INT	NOT NULL
primary_index_id	UNSIGNED INT	NOT NULL
match_type	TINYINT	NOT NULL
check_on_commit	CHAR(1)	NOT NULL
nulls	CHAR(1)	NOT NULL

foreign_table_id　外部テーブルのテーブル番号。

foreign_index_id　外部キーのインデックス番号。

primary_table_id　プライマリ・テーブルのテーブル番号。

primary_index_id　プライマリ・キーのインデックス番号。

match_type　制約と一致する型。一致する型には次の型があります。

値	一致する型
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」 460 ページの MATCH 句を参照してください。

check_on_commit (Y/N)　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。

nulls (Y/N)　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。

基礎となるシステム・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) は SYS.ISYSIDX (table_id, index_id) を参照

FOREIGN KEY (primary_table_id, primary_index_id) は SYS.ISYSIDX (table_id, index_id) を参照

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、グループとメンバの多対多の関係を示します。グループには複数のメンバがあり、あるユーザは複数のグループのメンバになれます。このビューの基礎となるシステム・テーブルは ISYSGROUP です。

カラム

カラム名	カラム型	カラム制約
group_id	UNSIGNED INT	NOT NULL
group_member	UNSIGNED INT	NOT NULL

group_id　グループのユーザ番号。

group_member　メンバのユーザ番号。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (group_id, group_member)

FOREIGN KEY (group_id) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (group_member) は SYS.ISYSUSER (user_id) を参照

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。このビューの基礎となるシステム・テーブルは ISYSHISTORY です。

カラム

カラム名	カラム型	カラム制約
operation	CHAR(128)	NOT NULL
object_id	UNSIGNED INT	NOT NULL
sub_operation	CHAR(128)	NOT NULL
version	CHAR(128)	NOT NULL
platform	CHAR(128)	NOT NULL
first_time	TIMESTAMP	NOT NULL
last_time	TIMESTAMP	NOT NULL
details	LONG VARCHAR	

operation　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 文または ALTER DATABASE

RESTORE DEFAULT CALIBRATION 文のどちらかの実行について、2 番目から最後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 文または ALTER DATABASE RESTORE DEFAULT CALIBRATION 文のどちらかの実行について、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・サーバのバージョンに関する情報。

object_id 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、SYSFILE システム・ビューで定義されている DB 領域の file_id です。

sub_operation 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 オペレーションの実行に使用されるデータベース・サーバのバージョンとビルド番号。

platform 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 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 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 このカラムは、データベース・サーバ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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (operation, object_id, version, platform)

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。このビューの基礎となるシステム・テーブルは ISYSIDX です。

カラム

カラム名	カラム型	カラム制約
table_id	UNSIGNED INT	NOT NULL

カラム名	カラム型	カラム制約
index_id	UNSIGNED INT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
phys_index_id	UNSIGNED INT	
file_id	SMALLINT	NOT NULL
index_category	TINYINT	NOT NULL
"unique"	TINYINT	NOT NULL
index_name	CHAR(128)	NOT NULL
not_enforced	CHAR(1)	NOT NULL

table_id　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　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　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　論理インデックスの実装に使用する基礎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。「SYSPHYSIDX システム・ビュー」 806 ページを参照してください。

file_id　インデックスを含むファイルの ID。この値は SYSFILE システム・ビューのエントリに対応します。「SYSFILE システム・ビュー」 792 ページを参照してください。

index_category　インデックスの型。次のような値があります。

値	インデックス・タイプ
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)

"unique"　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　インデックスの名前。

not_enforced　システムでのみ使用。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_id, index_id)

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (table_id, phys_index_id) は SYS.ISYSPHYSIDX (table_id, phys_index_id) を参照

参照

- 「SYSINDEXES システム・ビュー」 798 ページ
- 「SYSPHYSIDX システム・ビュー」 806 ページ
- 「SYSFILE システム・ビュー」 792 ページ

SYSINDEXES 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・ビューに含まれるインデックスの 1 カラムに関する記述です。このビューの基礎となるシステム・テーブルは ISYSIDXCOL です。

カラム

カラム名	カラム型	カラム制約
table_id	UNSIGNED INT	NOT NULL
index_id	UNSIGNED INT	NOT NULL
sequence	SMALLINT	NOT NULL
column_id	UNSIGNED INT	NOT NULL
"order"	CHAR(1)	NOT NULL
primary_column_id	UNSIGNED INT	

table_id　インデックスが適用されるテーブルを識別します。

index_id　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　インデックス中の各カラムには、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中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　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 (A/D)　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。

primary_column_id　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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_id, index_id, column_id)

FOREIGN KEY (table_id, index_id) は SYS.ISYSIDX (table_id, index_id) を参照

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

参照

- 「SYSIDX システム・ビュー」 796 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。このビューの基礎となるシステム・テーブルは ISYSJAR です。

カラム

カラム名	カラム型	カラム制約
jar_id	INTEGER	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
creator	UNSIGNED INT	NOT NULL
jar_name	LONG VARCHAR	NOT NULL
jar_file	LONG VARCHAR	
update_time	TIMESTAMP	NOT NULL

jar_id　JAR ファイルを識別するユニークな番号。

object_id　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　JAR ファイルの作成者のユーザ番号。

jar_name　JAR ファイルの名前。

jar_file　データベース内の JAR ファイルの外部ファイル名。

update_time　JAR ファイルが最後に更新された時刻。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (jar_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

参照

- 「SYSJARCOMPONENT システム・ビュー」 800 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ています。このビューの基礎となるシステム・テーブルは ISYSJARCOMPONENT です。

カラム

カラム名	カラム型	カラム制約
component_id	INTEGER	NOT NULL
jar_id	INTEGER	
component_name	LONG VARCHAR	
component_type	CHAR(1)	
contents	LONG BINARY	

component_id　コンポーネントの ID を含むプライマリ・キー。

jar_id　JAR の ID 番号を含むフィールド。

component_name　コンポーネント名。

component_type　コンポーネントの型。

contents　JAR ファイルのバイト・コード。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (component_id)

FOREIGN KEY (jar_id) は SYS.ISYSJAR (jar_id) を参照

参照

- 「SYSJAR システム・ビュー」 799 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 1 つの Java クラスに関する記述です。このビューの基礎となるシステム・テーブルは ISYSJAVACLASS です。

カラム

カラム名	カラム型	カラム制約
class_id	INTEGER	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
creator	UNSIGNED INT	NOT NULL

カラム名	カラム型	カラム制約
jar_id	INTEGER	
class_name	LONG VARCHAR	NOT NULL
public	CHAR(1)	NOT NULL
component_id	INTEGER	
update_time	TIMESTAMP	NOT NULL
class_descriptor	LONG BINARY	

class_id　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。

object_id　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　クラスの作成者のユーザ番号。

jar_id　クラスがある JAR ファイルの ID。

class_name　Java クラスの名前。

public　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　SYSJARCOMPONENT システム・ビューのコンポーネントの ID。

update_time　クラスが最後に更新された時刻。

class_descriptor　使用されません。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (class_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (creator) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (component_id) は SYS.ISYSJARCOMPONENT (component_id) を参照

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 権限を持つユーザだけが表示できます。このビューの基礎となるシステム・テーブルは ISYSLOGINMAP です。

カラム

カラム名	カラム型	カラム制約
login_mode	TINYINT	NOT NULL
login_id	VARCHAR(1024)	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
database_uid	UNSIGNED INT	NOT NULL

login_mode　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。

login_id　database_uid にマッピングする統合化ログインのユーザ・プロファイル名または Kerberos 原則。

object_id　ユニークな識別子。ユーザ ID とデータベースのユーザ ID 間のマッピングごとに 1 つ。

database_uid　ログイン ID をマッピングするデータベース・ユーザ ID。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (login_mode, login_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (database_uid) は SYS.ISYSUSER (user_id) を参照

SYSMVOPTION システム・ビュー

SYSMVOPTION システム・ビューの各ローは、実体化ビュー (Materialized View) を作成した時点のオプション値の記述です。ただし、この記述には、オプションのオプション名は含まれません。このビューの基礎となるシステム・テーブルは ISYSMVOPTION です。

カラム

カラム名	カラム型	カラム制約
view_object_id	UNSIGNED BIGINT	NOT NULL
option_id	UNSIGNED INT	NOT NULL
option_value	LONG VARCHAR	NOT NULL

view_object_id　実体化ビュー (Materialized View) のオブジェクト ID。

option_id　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　実体化ビュー (Materialized View) を作成した時点のオプション値。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (view_object_id, option_id)

FOREIGN KEY (view_object_id) は SYS.ISYSOBJECT (object_id) を参照

FOREIGN KEY (option_id) は SYS.ISYSMVOPTIONNAME (option_id) を参照

参照

- 「SYSMVOPTIONNAME システム・ビュー」 803 ページ

SYSMVOPTIONNAME システム・ビュー

SYSMVOPTIONNAME システム・ビューの各ローには、SYSMVOPTION システム・ビューで定義されているオプションの名前が含まれます。このビューの基礎となるシステム・テーブルは ISYSMVOPTIONNAME です。

カラム

カラム名	カラム型	カラム制約
option_id	UNSIGNED INT	NOT NULL
option_name	CHAR(128)	NOT NULL

option_id　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　オプションの名前。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (option_id)

参照

- 「SYSMVOPTION システム・ビュー」 802 ページ

SYSOBJECT システム・ビュー

SYSOBJECT システム・ビューの各ローにはデータベース・オブジェクトを記述します。このビューの基礎となるシステム・テーブルは ISYSOBJECT です。

カラム

カラム名	カラム型	カラム制約
object_id	UNSIGNED BIGINT	NOT NULL
status	TINYINT	NOT NULL
object_type	TINYINT	NOT NULL

カラム名	カラム型	カラム制約
creation_time	TIMESTAMP	NOT NULL

object_id　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　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　オブジェクトの種類。次のような値があります。

値	意味
1	テーブル
2	ビュー
3	実体化ビュー (Materialized View)
4	カラム
5	インデックス
6	プロシージャ
7	トリガ
8	イベント
9	ユーザ
10	パブリケーション
11	リモート型
12	ログインのマッピング
13	JAR
14	Java クラス
16	サービス

creation_time　オブジェクトを作成した日付と時刻。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (object_id)

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 ID PUBLIC に対する設定は、自分の設定を持たないユーザが使うデフォルトの設定になります。このビューの基礎となるシステム・テーブルは ISYSOPTION です。

カラム

カラム名	カラム型	カラム制約
user_id	UNSIGNED INT	NOT NULL
"option"	CHAR(128)	NOT NULL
"setting"	LONG VARCHAR	NOT NULL

user_id　このオプション設定が適用されるユーザ番号。

option　オプションの名前。

setting　現在のオプションの設定。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (user_id, "option")

FOREIGN KEY (user_id) は SYS.ISYSUSER (user_id) を参照

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。このビューの基礎となるシステム・テーブルは ISYSOPTSTAT です。

カラム

カラム名	カラム型	カラム制約
stat_id	UNSIGNED INT	NOT NULL
group_id	UNSIGNED INT	NOT NULL
format_id	SMALLINT	NOT NULL

カラム名	カラム型	カラム制約
data	LONG BINARY	

stat_id　システムでのみ使用。

group_id　システムでのみ使用。

format_id　システムでのみ使用。

data　システムでのみ使用。

基礎となるシステム・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。このビューの基礎となるシステム・テーブルは ISYSPHYSIDX です。

カラム

カラム名	カラム型	カラム制約
table_id	UNSIGNED INT	NOT NULL
phys_index_id	UNSIGNED INT	NOT NULL
root	INTEGER	NOT NULL
key_value_count	UNSIGNED INT	NOT NULL
leaf_page_count	UNSIGNED INT	NOT NULL
depth	UNSIGNED SMALLINT	NOT NULL
max_key_distance	UNSIGNED INT	NOT NULL
seq_transitions	UNSIGNED INT	NOT NULL
rand_transitions	UNSIGNED INT	NOT NULL
rand_distance	UNSIGNED INT	NOT NULL
allocation_bitmap	LONG VARBIT	
long_value_bitmap	LONG VARBIT	

table_id　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　テーブル内の物理インデックスのユニークな番号。

root　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　インデックス内の個別のキー値の数。

leaf_page_count　リーフ・インデックス・ページの数。

depth　物理インデックスの深さ (レベル数)。

max_key_distance　システムでのみ使用。

seq_transitions　システムでのみ使用。

rand_transitions　システムでのみ使用。

rand_distance　システムでのみ使用。

allocation_bitmap　システムでのみ使用。

long_value_bitmap　システムでのみ使用。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_id, phys_index_id)

参照

- 「SYSINDEXES システム・ビュー」 798 ページ
- 「SYSIDX システム・ビュー」 796 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 ISYSPROCEDURE です。

カラム

カラム名	カラム型	カラム制約
proc_id	UNSIGNED INT	NOT NULL
creator	UNSIGNED INT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
proc_name	CHAR(128)	NOT NULL
proc_defn	LONG VARCHAR	
remarks	LONG VARCHAR	
replicate	CHAR(1)	NOT NULL
srvid	UNSIGNED INT	

カラム名	カラム型	カラム制約
source	LONG VARCHAR	
avg_num_rows	FLOAT	
avg_cost	FLOAT	
stats	LONG BINARY	

proc_id　各プロシージャにはユニークな番号 (プロシージャ番号) が割り当てられます。

creator　プロシージャの所有者。

object_id　データベースのプロシージャをユニークに識別するプロシージャの内部 ID。

proc_name　プロシージャ名。1 人の作成者が、同じ名前のプロシージャを 2 つ持つことはできません。

proc_defn　プロシージャの定義。

remarks　プロシージャに関する注記。ISYSREMARK システム・テーブル内に格納されている値。

replicate　(Y/N) プロシージャが Replication Server インストール環境のプライマリ・データ・ソースであるかどうかを表します。

srvid　プロシージャがリモート・データベース・サーバ上のプロシージャのプロキシである場合は、そのリモート・サーバを表します。

source　回避されたプロシージャのソース。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　プロシージャが 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　プロシージャが 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。

stats　プロシージャが 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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (proc_id)

FOREIGN KEY (srvid) は SYS.ISYSSERVER (srvid) を参照

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (creator) は SYS.ISYSUSER (user_id) を参照

UNIQUE (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に対するパラメータの記述です。このビューの基礎となるシステム・テーブルは ISYSPROCPARM です。

カラム

カラム名	カラム型	カラム制約
proc_id	UNSIGNED INT	NOT NULL
parm_id	SMALLINT	NOT NULL
parm_type	SMALLINT	NOT NULL
parm_mode_in	CHAR(1)	NOT NULL
parm_mode_out	CHAR(1)	NOT NULL
domain_id	SMALLINT	NOT NULL
width	UNSIGNED INT	NOT NULL
scale	SMALLINT	NOT NULL
user_type	SMALLINT	
parm_name	CHAR(128)	NOT NULL
"default"	LONG VARCHAR	

proc_id　このパラメータが属するプロシージャをユニークに識別します。

parm_id　各プロシージャは、パラメータに 1 から順に番号を付けます。パラメータ番号の順は、定義された順になっています。

関数の場合、最初のパラメータの内容は関数名であり、その関数の戻り値を表します。

parm_type　パラメータは、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 (変数)
- **1**　結果変数 – 結果セットを返すプロシージャで使用
- **2**　SQLSTATE エラー値
- **3**　SQLCODE エラー値
- **4**　関数からの戻り値

parm_mode_in (Y/N)　このパラメータが、プロシージャ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)。

parm_mode_out (Y/N)　このパラメータが、プロシージャからの値 (IN または INOUT パラメータ) と RESULT 句のカラムのどちらを返すかを示します。

domain_id　SYSDOMAIN システム・ビューにリストされたデータ型番号から、パラメータのデータ型を識別します。

width　文字列パラメータでは長さ、数値パラメータでは精度、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　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　パラメータのユーザ型 (適用できる場合)。

parm_name　プロシージャ・パラメータの名前。

"default"　パラメータのデフォルト値。参照情報としてのみ表示されます。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (proc_id, parm_id)

FOREIGN KEY (proc_id) は SYS.ISYSPROCEDURE (proc_id) を参照

FOREIGN KEY (domain_id) は SYS.ISYSDOMAIN (domain_id) を参照

FOREIGN KEY (user_type) は SYS.ISYSUSERTYPE (type_id) を参照

SYSPROCPERM システム・ビュー

SYSPROCPERM システム・ビューの各ローは、1 つのプロシージャを実行するための、ユーザに付与された 1 つのパーミッションに関する記述です。パーミッションを付与されたユーザだけがプロシージャを実行できます。このビューの基礎となるシステム・テーブルは ISYSPROCPERM です。

カラム

カラム名	カラム型	カラム制約
proc_id	UNSIGNED INT	NOT NULL
grantee	UNSIGNED INT	NOT NULL

proc_id　プロシージャ番号は、パーミッションが付与されたプロシージャをユニークに特定します。

grantee　パーミッションを受けるユーザのユーザ番号。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (proc_id, grantee)

FOREIGN KEY (grantee) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (proc_id) は SYS.ISYSPROCEDURE (proc_id) を参照

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 ISYSPROXYTAB です。

カラム

カラム名	カラム型	カラム制約
table_object_id	UNSIGNED BIGINT	NOT NULL
existing_obj	CHAR(1)	
srvid	UNSIGNED INT	
remote_location	LONG VARCHAR	

table_object_id　プロキシ・テーブルのオブジェクト ID。

existing_obj　プロキシがリモート・サーバに既に存在しているかどうかを示します (Y または N)。

srvid　プロキシ・テーブルと関連付けられたリモート・サーバのユニークな ID。

remote_location　リモート・サーバに関するプロキシ・テーブルの位置。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_object_id)

FOREIGN KEY (table_object_id) は ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (srvid) は SYS.ISYSSERVER (srvid) を参照

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 ISYSPUBLICATION です。

カラム

カラム名	カラム型	カラム制約
publication_id	UNSIGNED INT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
creator	UNSIGNED INT	NOT NULL

カラム名	カラム型	カラム制約
publication_name	CHAR(128)	NOT NULL
remarks	LONG VARCHAR	
type	CHAR(1)	NOT NULL
sync_type	UNSIGNED INT	NOT NULL

publication_id　パブリケーションをユニークに識別する番号。

object_id　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の内部 ID。

creator　パブリケーションの所有者。

publication_name　パブリケーションの名前。

remarks　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　このカラムは使用されません。

sync_type　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。

- **logscan**　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。
- **scripted upload**　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を使用してユーザが定義します。ストアド・プロシージャに関する情報は、ISYSSYNCSCRIPT システム・テーブルに格納されます。
- **download only**　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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (publication_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (creator) は SYS.ISYSUSER (user_id) を参照

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」 822 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) 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 ISYSREMARK です。

カラム

カラム	データ型	カラム制約
object_id	UNSIGNED BIGINT	NOT NULL
remarks	LONG VARCHAR	NOT NULL

object_id　関連する注釈があるオブジェクトの内部 ID。

remarks　オブジェクトと関連付けられた注釈またはコメント。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

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・パラメータの値に関する記述です。このビューの基礎となるシステム・テーブルは ISYSREMOTEOPTION です。

カラム

カラム	データ型	カラム制約
option_id	UNSIGNED INT	NOT NULL
user_id	UNSIGNED INT	NOT NULL
"setting"	VARCHAR(255)	NOT NULL

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。

option_id　メッセージ・リンク・パラメータの ID 番号。

user_id　パラメータが設定されているユーザ ID。

"setting"　メッセージ・リンク・パラメータの値。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (option_id, user_id)

FOREIGN KEY (option_id) は SYS.ISYSREMOTEOPTIONTYPE (option_id) を参照

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・パラメータのいずれかに関する記述です。このビューの基礎となるシステム・テーブルは ISYSREMOTEOPTIONTYPE です。

カラム

カラム	データ型	カラム制約
option_id	UNSIGNED INT	NOT NULL
type_id	SMALLINT	NOT NULL
"option"	VARCHAR(128)	NOT NULL

option_id　メッセージ・リンク・パラメータの ID 番号。

type_id　このパラメータを使用するメッセージ・タイプの ID 番号。

"option"　メッセージ・リンク・パラメータの名前。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (option_id)

FOREIGN KEY (type_id) は SYS.ISYSREMOTETYPE (type_id) を参照

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。このビューの基礎となるシステム・テーブルは ISYSREMOTETYPE です。

カラム

カラム名	カラム型	カラム制約
type_id	SMALLINT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
type_name	CHAR(128)	NOT NULL
publisher_address	LONG VARCHAR	NOT NULL
remarks	LONG VARCHAR	

type_id　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　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　SQL Remote がサポートするメッセージ・システムの名前。

publisher_address　リモート・データベース・パブリッシャのアドレス。

remarks　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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (type_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

UNIQUE (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 ID (サブスクライバ) を記述します。そのユーザに送信された SQL Remote メッセージと、そのユーザ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。このビューの基礎となるシステム・テーブルは ISYSREMOTEUSER です。

カラム

カラム名	カラム型	カラム制約
user_id	UNSIGNED INT	NOT NULL
consolidate	CHAR(1)	NOT NULL
type_id	SMALLINT	NOT NULL
address	LONG VARCHAR	NOT NULL
frequency	CHAR(1)	NOT NULL
send_time	TIME	
log_send	UNSIGNED BIGINT	NOT NULL
time_sent	TIMESTAMP	
log_sent	UNSIGNED BIGINT	NOT NULL
confirm_sent	UNSIGNED BIGINT	NOT NULL
send_count	INTEGER	NOT NULL
resend_count	INTEGER	NOT NULL
time_received	TIMESTAMP	
log_received	UNSIGNED BIGINT	NOT NULL
confirm_received	UNSIGNED BIGINT	

カラム名	カラム型	カラム制約
receive_count	INTEGER	NOT NULL
rereceive_count	INTEGER	NOT NULL

user_id REMOTE パーミッションを持つユーザのユーザ番号。

consolidate (Y/N) ユーザに 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 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 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 SQL Remote メッセージが送信される頻度。

send_time 次にメッセージがこのユーザへ送信される時刻。

log_send メッセージは、log_send が log_sent よりも大きいサブスクライバだけに送信されます。

time_sent このサブスクライバに、最後にメッセージが送信された時刻。

log_sent 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 このサブスクライバ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 SQL Remote メッセージが送信された回数。

resend_count メッセージがサブスクライバ・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。

time_received このサブスクライバから、最後にメッセージが受信された時刻。

log_received 現在のデータベースで最後に受信された操作の、サブスクライバ・データベースのログ・オフセット。

confirm_received 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・データベース内のログ・オフセット。

receive_count メッセージが受信された回数。

rereceive_count 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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (user_id)

FOREIGN KEY (user_id) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (type_id) は SYS.ISYSREMOTETYPE (type_id) を参照

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。このビューの基礎となるシステム・テーブルは ISYSSCHEDULE です。

カラム

カラム名	カラム型	カラム制約
event_id	UNSIGNED INT	NOT NULL
sched_name	VARCHAR(128)	NOT NULL
recurring	TINYINT	NOT NULL
start_time	TIME	NOT NULL
stop_time	TIME	
start_date	DATE	
days_of_week	TINYINT	
days_of_month	UNSIGNED INT	
interval_units	CHAR(10)	
interval_amt	INTEGER	

event_ id　各イベントに割り当てられたユニークな番号。

sched_name　イベントのスケジュールと関連付けられた名前。

recurring (0/1)　スケジュールを反復するかどうかを示します。

start_time　スケジュールの起動時刻。

stop_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　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　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　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

interval_units　EVERY で指定される間隔単位。

- HH = 時間
- NN = 分
- SS = 秒

interval_amt　EVERY で指定される期間。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (event_id, sched_name)

FOREIGN KEY (event_id) は SYS.ISYSEVENT (event_id) を参照

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 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、このビューの基礎となるテーブルになります。

カラム

カラム名	カラム型	カラム制約
srvid	UNSIGNED INT	NOT NULL
srvname	VARCHAR(128)	NOT NULL
srvclass	LONG VARCHAR	NOT NULL
srvinfo	LONG VARCHAR	
srvreadonly	CHAR(1)	NOT NULL

srvid　リモート・サーバの識別子。

srvname　リモート・サーバの名前。

srvclass　CREATE SERVER 文で指定されたサーバ・クラス。

srvinfo　サーバ情報。

srvreadonly　サーバが読み込み専用の場合は Y、それ以外の場合は N。

基礎となるシステム・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。このビューの基礎となるシステム・テーブルは ISYSSOURCE です。

カラム

カラム名	カラム型	カラム制約
object_id	UNSIGNED BIGINT	NOT NULL
source	LONG VARCHAR	NOT NULL

object_id　ソース・コードが定義されているオブジェクトの内部 ID。

source　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。詳細については、「preserve_source_format オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

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。このビューの基礎となるシステム・テーブルは ISYSSQLSERVERTYPE です。

カラム

カラム名	カラム型	カラム制約
ss_user_type	SMALLINT	NOT NULL

カラム名	カラム型	カラム制約
ss_domain_id	SMALLINT	NOT NULL
ss_type_name	VARCHAR(30)	NOT NULL
primary_sa_domain_id	SMALLINT	NOT NULL
primary_sa_user_type	SMALLINT	

ss_user_type Adaptive Server Enterprise ユーザ型。

ss_domain_id Adaptive Server Enterprise のドメイン ID。

ss_type_name Adaptive Server Enterprise 型名。

primary_sa_domain_id 対応する SQL Anywhere のプライマリ・ドメイン ID。

primary_sa_user_type 対応する SQL Anywhere のプライマリ・ユーザ・タイプ。

基礎となるシステム・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 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 ISYSSUBSCRIPTION です。

カラム

カラム名	カラム型	カラム制約
publication_id	UNSIGNED INT	NOT NULL
user_id	UNSIGNED INT	NOT NULL
subscribe_by	CHAR(128)	NOT NULL
created	UNSIGNED BIGINT	NOT NULL
started	UNSIGNED BIGINT	

publication_id ユーザ 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 パブリケーションに対して送信されたユーザの ID。

subscribe_by サブスクリプション用の SUBSCRIBE BY 式がある場合、その値。

created 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 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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (publication_id, user_id, subscribe_by)

FOREIGN KEY (publication_id) は SYS.ISYSPUBLICATION (publication_id) を参照

FOREIGN KEY (user_id) は SYS.ISYSUSER (user_id) を参照

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。このビューの基礎となるシステム・テーブルは ISYSSYNC です。

カラム

カラム名	カラム型	カラム制約
sync_id	UNSIGNED INT	NOT NULL
type	CHAR(1)	NOT NULL
publication_id	UNSIGNED INT	
progress	UNSIGNED BIGINT	
site_name	CHAR(128)	
"option"	LONG VARCHAR	
server_connect	LONG VARCHAR	
server_conn_type	LONG VARCHAR	
last_download_time	TIMESTAMP	
last_upload_time	TIMESTAMP	NOT NULL
created	UNSIGNED BIGINT	
log_sent	UNSIGNED BIGINT	
generation_number	INTEGER	NOT NULL
extended_state	VARCHAR(1024)	NOT NULL

sync_id　ユニークにローを識別する番号。

type　同期オブジェクトのタイプ : D は定義、T はテンプレート、S はサイトを意味します。

publication_id　SYSPUBLICATION システム・ビューに入っている publication_id。

progress　最後に成功したアップロードのログ・オフセット。

site_name　Mobile Link ユーザ名。

"option"　同期オプション。

server_connect　Mobile Link サーバのアドレスまたは URL。

server_conn_type　同期時に使用する、TCP/IP などの通信プロトコル。

last_download_time　最後に Mobile Link サーバからダウンロード・ストリームを受信した時刻。

last_upload_time　最後に情報のアップロードが成功した時刻 (Mobile Link サーバで測定)。デフォルトは jan-1-1990 です。

created　サブスクリプションが作成されたログ・オフセット。

log_sent　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　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　内部用に予約されています。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (sync_id)

FOREIGN KEY (publication_id) は SYS.ISYSPUBLICATION (publication_id) を参照

UNIQUE (publication_id, site_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。ユーザが読むことができる形式で表示するには、「SYSSYNCSCRIPTS 統合ビュー」 849 ページを参照してください。

スクリプト化したアップロードを使用するパブリケーションの詳細については、「SYSPUBLICATION システム・ビュー」 811 ページを参照してください。ストアド・プロシージャの定義については、「SYSPROCEDURE システム・ビュー」 807 ページを参照してください。

このビューの基礎となるシステム・テーブルは ISYSSYNCSCRIPT です。

カラム名	カラム型	カラム制約
pub_object_id	UNSIGNED BIGINT	NOT NULL
table_object_id	UNSIGNED BIGINT	NOT NULL

カラム名	カラム型	カラム制約
type	UNSIGNED INT	NOT NULL
proc_object_id	UNSIGNED BIGINT	NOT NULL

カラム

pub_object_id　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　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　アップロード・プロシージャのタイプ。

proc_object_id　パブリケーションに使用するストアド・プロシージャのオブジェクト ID。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (pub_object_id, table_object_id, type)

FOREIGN KEY (pub_object_id) は SYS.ISYSOBJECT (object_id) を参照

FOREIGN KEY (table_object_id) は SYS.ISYSOBJECT (object_id) を参照

FOREIGN KEY (proc_object_id) は SYS.ISYSOBJECT (object_id) を参照

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』
- 「SYSPUBLICATION システム・ビュー」 811 ページ
- 「SYSPROCEDURE システム・ビュー」 807 ページ
- 「SYSSYNCSCRIPTS 統合ビュー」 849 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 1 つのテーブルに関する記述です。ビューの追加情報が SYSVIEW システム・ビューにあります。このビューの基礎となるシステム・テーブルは ISYSTAB です。

カラム

カラム名	カラム型	カラム制約
table_id	UNSIGNED INT	NOT NULL
file_id	SMALLINT	NOT NULL
count	UNSIGNED BIGINT	NOT NULL
creator	UNSIGNED INT	NOT NULL
table_page_count	INTEGER	NOT NULL

カラム名	カラム型	カラム制約
ext_page_count	INTEGER	NOT NULL
commit_action	INTEGER	NOT NULL
share_type	INTEGER	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
last_modified_at	TIMESTAMP	NOT NULL
table_name	CHAR(128)	NOT NULL
table_type	TINYINT	NOT NULL
replicate	CHAR(1)	NOT NULL
server_type	TINYINT	NOT NULL
tab_page_list	LONG VARBIT	NULL
ext_page_list	LONG VARBIT	NULL
pct_free	UNSIGNED INT	NULL
clustered_index_id	UNSIGNED INT	NULL
encrypted	CHAR(1)	NOT NULL
table_type_str	CHAR(8)	NOT NULL

table_id　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。

file_id　テーブルを含む DB 領域を示す値。

count　テーブルまたは実体化ビュー (Materialized View) 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。SQL Anywhere はこの番号を使ってデータベース・アクセスを最適化します。非実体化ビュー (Non-materialized View) 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　このユーザ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　基礎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　このテーブルで使用される拡張ページの総数。

commit_action　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　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　テーブルのオブジェクト ID。

last_modified_at　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。

table_name　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　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。

値	テーブルのタイプ
1	ベース・テーブル
2	実体化ビュー (Materialized View)
3	グローバル・テンポラリ・テーブル
21	ビュー

replicate　基礎となるテーブルが Replication Server インストール環境のプライマリ・データ・ソースであるかどうかを示す値。

server_type　基礎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。

値	場所
1	ローカル・サーバ (SQL Anywhere)
3	リモート・サーバ

tab_page_list　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)。これは内部でのみ使用されます。

ext_page_list　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)。これは内部でのみ使用されます。

pct_free　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　テーブルのクラスタ化されたインデックスの ID。クラスタ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　テーブルまたは実体化ビュー (Materialized View) を暗号化するかどうかを示す値 (Y または N)。

table_type_str　table_type の読み取り可能な値。次のような値があります。

値	テーブルのタイプ
BASE	ベース・テーブル
MAT VIEW	実体化ビュー (Materialized View)
GBL TEMP	グローバル・テンポラリ・テーブル
VIEW	ビュー

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_id)

FOREIGN KEY (file_id) は SYS.ISYSFILE (file_id) を参照

FOREIGN KEY (creator) references SYS.ISYSUSER (user_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

UNIQUE (table_name, creator)

UNIQUE (table_name)

参照

- 「SYSVIEW システム・ビュー」 835 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。このビューの基礎となるシステム・テーブルは ISYSTABCOL です。

カラム

カラム名	カラム型	カラム制約
table_id	UNSIGNED INT	NOT NULL
column_id	UNSIGNED INT	NOT NULL
domain_id	SMALLINT	NOT NULL
nulls	CHAR(1)	NOT NULL
width	UNSIGNED INT	NOT NULL
scale	SMALLINT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
max_identity	BIGINT	NOT NULL

カラム名	カラム型	カラム制約
column_name	CHAR(128)	NOT NULL
"default"	LONG VARCHAR	
user_type	SMALLINT	
column_type	CHAR(1)	NOT NULL
compressed	TINYINT	NOT NULL
collect_stats	TINYINT	NOT NULL
inline_max	SMALLINT	
inline_long	SMALLINT	
lob_index	TINYINT	

table_id　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　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。この番号は、ORDER BY 句を指定していない場合、SELECT コマンドから返されるカラムの順序を決定します。

```
SELECT * FROM table
```

domain_id　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 (Y/N)　NULL 値をカラムに許可するかどうかを指定します。

width　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　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　テーブル・カラムのオブジェクト ID。

max_identity　AUTOINCREMENT、IDENTITY、または GLOBAL AUTOINCREMENT カラムの場合、カラムの最大値。

column_name　カラム名。

default　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　ユーザ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　カラムのタイプ (C=計算済みカラム、R=他のカラム)。

compressed　このカラムを圧縮形式で格納するかどうか。

collect_stats　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。

inline_max　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。

NOT NULL の 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 句の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

inline_long　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。

NOT NULL の inline_long 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。PREFIX 句の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

lob_index　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。

NOT NULL の lob_index 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。[NO] INDEX 句の詳細については、「CREATE TABLE 文」 460 ページを参照してください。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (table_id, column_id)

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

FOREIGN KEY (domain_id) は SYS.ISYSDOMAIN (domain_id) を参照

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (user_type) は SYS.ISYSUSERTYPE (type_id) を参照

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 ID (「**grantor**」)、そしてパーミッションを与えられるユーザ ID (「**grantee**」) に対応します。このビューの基礎となるシステム・テーブルは ISYSTABLEPERM です。

カラム

カラム名	カラム型	カラム制約
stable_id	UNSIGNED INT	NOT NULL
grantee	UNSIGNED INT	NOT NULL

カラム名	カラム型	カラム制約
grantor	UNSIGNED INT	NOT NULL
selectauth	CHAR(1)	NOT NULL
insertauth	CHAR(1)	NOT NULL
deleteauth	CHAR(1)	NOT NULL
updateauth	CHAR(1)	NOT NULL
updatecols	CHAR(1)	NOT NULL
alterauth	CHAR(1)	NOT NULL
referenceauth	CHAR(1)	NOT NULL

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に付与できます。「GRANT 文」 565 ページを参照してください。

パーミッション
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。

stable_id 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 パーミッションを受けるユーザ ID のユーザ番号。

grantor パーミッションを付与するユーザ ID のユーザ番号。

selectauth (Y/N/G) SELECT パーミッションが付与されたかどうかを示します。

insertauth (Y/N/G) INSERT パーミッションが付与されたかどうかを示します。

deleteauth (Y/N/G) DELETE パーミッションが付与されたかどうかを示します。

updateauth (Y/N/G) UPDATE パーミッションが、テーブル中のすべてのカラムに付与されたかどうかを示します。

updatecols (Y/N) UPDATE パーミッションが、基礎となるテーブル中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 (Y/N/G) ALTER パーミッションが付与されたかどうかを示します。

referenceauth (Y/N/G)　REFERENCES パーミッションが付与されたかどうかを示します。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (stable_id, grantee, grantor)

FOREIGN KEY (stable_id) は SYS.ISYSTAB (table_id) を参照

FOREIGN KEY (ttable_id) は SYS.ISYSTAB (table_id) を参照

FOREIGN KEY (grantor) は SYS.ISYSUSER (user_id) を参照

FOREIGN KEY (grantee) は SYS.ISYSUSER (user_id) を参照

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に関する記述です。このビューには、参照トリガ・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も含まれます (たとえば、ON DELETE CASCADE)。このビューの基礎となるシステム・テーブルは ISYSTRIGGER です。

カラム

カラム名	カラム型	カラム制約
trigger_id	UNSIGNED INT	NOT NULL
table_id	UNSIGNED INT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
event	CHAR(1)	NOT NULL
trigger_time	CHAR(1)	NOT NULL
trigger_order	SMALLINT	
foreign_table_id	UNSIGNED INT	
foreign_key_id	UNSIGNED INT	
referential_action	CHAR(1)	
trigger_name	CHAR(128)	
trigger_defn	LONG VARCHAR	NOT NULL
remarks	LONG VARCHAR	
source	LONG VARCHAR	

trigger_id　各トリガにはユニークな番号 (トリガ番号) が割り当てられます。

table_id　トリガが所属するテーブルのテーブル ID。

event　トリガを起動する 1 つまたは複数のイベント。この 1 文字の値は、トリガを作成したときに指定したトリガ・イベントに対応します。

- **A**　INSERT、DELETE
- **B**　INSERT、UPDATE
- **C**　UPDATE COLUMNS
- **D**　DELETE
- **E**　DELETE、UPDATE
- **I**　INSERT
- **U**　UPDATE
- **M**　INSERT、DELETE、UPDATE

trigger_time　トリガが起動される時刻。この 1 文字の値は、トリガを作成したときに指定したトリガ起動時刻に対応します。

- **A**　AFTER (ロー・レベルのトリガ)
- **B**　BEFORE (ロー・レベルのトリガ)
- **R**　RESOLVE
- **S**　AFTER (文レベルのトリガ)

trigger_order　トリガが起動される順番。同じ型 (insert、update、delete) のトリガが同時 (before、after) に起動されるときに使われます。

foreign_table_id　参照トリガ・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テーブル番号。

foreign_key_id　foreign_table_id が参照するテーブルの外部キーの外部キー番号。

referential_action　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　トリガ名。1 つのテーブルには、同じ名前のトリガは複数存在できません。

trigger_defn　トリガを作成するのに使ったコマンド。

remarks　トリガに関する注記。ISYSREMARK システム・テーブル内に格納されている値。

source　トリガの 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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (trigger_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

FOREIGN KEY (table_id) は SYS.ISYSTAB (table_id) を参照

FOREIGN KEY (foreign_table_id, foreign_key_id) は SYS.ISYSIDX (table_id, index_id) を参照

UNIQUE (table_id, event, trigger_time, trigger_order)

UNIQUE (trigger_name, table_id)

UNIQUE (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。このビューの基礎となるシステム・テーブルは ISYSTYPEMAP です。

カラム

カラム名	カラム型	カラム制約
ss_user_type	SMALLINT	NOT NULL
sa_domain_id	SMALLINT	NOT NULL
sa_user_type	SMALLINT	
nullable	CHAR(1)	

ss_user_type　Adaptive Server Enterprise ユーザ型を格納します。

sa_domain_id　対応する SQL Anywhere domain_id を格納します。

sa_user_type　対応する SQL Anywhere ユーザ型を格納します。

nullable　このフィールドには、型が NULL 値を許容するかどうかを記述します。

基礎となるシステム・テーブルに関する制約

FOREIGN KEY (sa_domain_id) は SYS.ISYSDOMAIN (domain_id) を参照

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 ISYSUSER です。

カラム

カラム名	カラム型	カラム制約
user_id	UNSIGNED INT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
user_name	CHAR(128)	NOT NULL
password	BINARY(128)	

user_id　システムのユーザをユニークに識別する番号。

object_id　データベースのユーザをユニークに識別するユーザの内部 ID。

user_name　ユーザのログイン名。

password　ユーザのパスワード。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (user_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

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 ID に付与された権限を記述します。このビューの基礎となるシステム・テーブルは、ISYSUSERAUTHORITY です。

カラム

カラム名	カラム型	カラム制約
user_id	UNSIGNED INT	NOT NULL
auth	VARCHAR(20)	NOT NULL

user_id　権限が所属するユーザの ID。

auth　ユーザに付与されている権限。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (user_id, auth)

FOREIGN KEY (user_id) は SYS.ISYSUSER (user_id) を参照

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定義メッセージが入っています。このビューの基礎となるシステム・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、このビューの基礎となるテーブルになります。

カラム

カラム名	カラム型	カラム制約
error	INTEGER	NOT NULL
uid	UNSIGNED INT	NOT NULL
description	VARCHAR(255)	NOT NULL
langid	SMALLINT	NOT NULL

error　エラー条件に対するユニークな識別番号。

uid　メッセージを定義するユーザ番号。

description　エラー条件に対応するメッセージ。

langid　予約。

基礎となるシステム・テーブルに関する制約

UNIQUE (error, langid)

SYSUSERTYPE システム・ビュー

SYSUSERTYPE システム・ビューの各ローは、ユーザ定義のデータ型を記述します。このビューの基礎となるシステム・テーブルは ISYSUSERTYPE です。

カラム

カラム名	カラム型	カラム制約
type_id	SMALLINT	NOT NULL
creator	UNSIGNED INT	NOT NULL
domain_id	SMALLINT	NOT NULL
nulls	CHAR(1)	NOT NULL
width	UNSIGNED INT	NOT NULL

カラム名	カラム型	カラム制約
scale	SMALLINT	NOT NULL
type_name	CHAR(128)	NOT NULL
"default"	LONG VARCHAR	
"check"	LONG VARCHAR	

type_id　ユーザ定義データ型のユニークな識別番号。

creator　データ型の所有者のユーザ番号。

domain_id　SYSDOMAIN システム・ビューにリストされたデータ型番号で示す、ユーザ定義データ型の元になるデータ型。

nulls　(Y/N/U) ユーザ定義データ型が null を許可するかどうかを示します。値 U は、null を指定可能かどうかが指定されていないことを示します。

width　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　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　データ型の名前。

"default"　データ型のデフォルト値。

"check"　データ型の CHECK 条件。

基礎となるシステム・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) は SYS.ISYSDOMAIN (domain_id) を参照

UNIQUE (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に関する記述です。ビューの追加情報が SYSTAB システム・ビューにあります。このビューの基礎となるシステム・テーブルは ISYSVIEW です。

実体化ビュー (Materialized View) の情報をより読みやすい形式にした sa_materialized_view_info システム・プロシージャも使用できます。「sa_materialized_view_info システム・プロシージャ」 919 ページを参照してください。

カラム

カラム名	カラム型	カラム制約
view_object_id	UNSIGNED BIGINT	NOT NULL
view_def	LONG VARCHAR	NOT NULL
mv_build_type	TINYINT	
mv_refresh_type	TINYINT	
mv_use_in_optimization	TINYINT	
mv_last_refreshed_at	TIMESTAMP	
mv_known_stale_at	TIMESTAMP	

view_object_id　ビューのオブジェクト ID。

view_def　ビューの定義 (クエリ仕様)。

mv_build_type　現在使われていません。

mv_refresh_type　現在使われていません。

mv_use_in_optimization　このカラムは実体化ビュー (Materialized View) 専用です。また、クエリの最適化時に実体化ビュー (Materialized View) を使用できるかどうかを示します (0=最適化で使用できません、1=最適化で使用できます)。「オプティマイザによる実体化ビュー (Materialized View) の使用の有効化と無効化」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

mv_last_refreshed_at　このカラムは、実体化ビュー (Materialized View) 専用です。また、最後の更新日時を示します。

mv_known_stale_at　このカラムは実体化ビュー (Materialized View) 専用です。実体化ビュー (Materialized View) が古くなったときを示します。この値は、基礎となる基本テーブルの 1 つが変更されたために検証されたときに合致します。0 の値は、ビューが新しいか、古いけれどもデータベース・サーバから古いと印が付けられていないこと (古くなつ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・プロシージャを使用して、実体化ビュー (Materialized View) のステータスを決定します。「sa_materialized_view_info システム・プロシージャ」 919 ページを参照してください。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (view_object_id)

FOREIGN KEY (view_object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

参照

- 「SYSTAB システム・ビュー」 823 ページ
- 「CREATE MATERIALIZED VIEW 文」 420 ページ
- 「REFRESH MATERIALIZED VIEW 文」 640 ページ
- 「CREATE VIEW 文」 482 ページ

SYSWEBSERVICE システム・ビュー

SYSWEBSERVICE システム・ビューの各ローは、Web サービスの説明を保持します。このビューの基礎となるシステム・テーブルは ISYSWEBSERVICE です。

カラム

カラム名	カラム型	カラム制約
service_id	UNSIGNED INT	NOT NULL
object_id	UNSIGNED BIGINT	NOT NULL
service_name	CHAR(128)	NOT NULL
service_type	VARCHAR(40)	NOT NULL
auth_required	CHAR(1)	NOT NULL
secure_required	CHAR(1)	NOT NULL
url_path	CHAR(1)	NOT NULL
user_id	UNSIGNED INT	
parameter	VARCHAR(250)	
statement	LONG VARCHAR	
remarks	LONG VARCHAR	

service_id　Web サービスをユニークに識別する番号。

object_id　Web サービスの ID。

service_name　Web サービスに割り当てられた名前。

service_type　サービスのタイプには、RAW、HTTP、XML、SOAP、DISH などがあります。

auth_required　(Y/N) すべての要求に、有効なユーザ名とパスワードが必要かどうかを示します。

secure_required　(Y/N) 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を示します。

url_path　URL の解釈を制御します。

user_id　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またはユーザ・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　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　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 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。

基礎となるシステム・テーブルに関する制約

PRIMARY KEY (service_id)

FOREIGN KEY (object_id) は SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL を参照

UNIQUE (service_name)

統合ビュー

統合ビューは、ユーザ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、基礎となるシステム・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。

SYSARTICLECOLS 統合ビュー

SYSARTICLECOLS ビューの各ローはアーティクルのカラムを識別します。

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"."SYSARTICLECOLS"
  as select p.publication_name,t.table_name,c.column_name from
    SYS.SYSARTICLECOL as ac join
    SYS.SYSPUBLICATION as p on p.publication_id = ac.publication_id join
    SYS.SYSTABLE as t on t.table_id = ac.table_id join
    SYS.SYSCOLUMN 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」 783 ページ
- 「SYSPUBLICATION システム・ビュー」 811 ページ
- 「SYSTABLE 互換ビュー (旧式)」 858 ページ

SYSARTICLES 統合ビュー

SYSARTICLES ビューの各ローは、パブリケーションのアーティクルを記述します。

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。

```
CREATE 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 a
      JOIN SYS.ISYSPUBLICATION p ON ( a.publication_id = p.publication_id )
      JOIN SYS.ISYSTAB t ON ( a.table_id = t.table_id )
      JOIN SYS.ISYSUSER u1 ON ( p.creator = u1.user_id )
      JOIN SYS.ISYSUSER u2 ON ( t.creator = u2.user_id )
```

参照

- 「SYSARTICLE システム・ビュー」 782 ページ
- 「SYSPUBLICATION システム・ビュー」 811 ページ
- 「SYSTAB システム・ビュー」 823 ページ
- 「SYSUSER システム・ビュー」 832 ページ

SYSCAPABILITIES 統合ビュー

SYSCAPABILITIES ビューの各ローには、機能を記述します。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 t1.capid,t1.srvid,t2.capname,t1.capvalue from
    SYS.ISYSCAPABILITY as t1 join
    SYS.ISYSCAPABILITYNAME as t2 on t1.capid = t2.capid
```

参照

- 「SYSCAPABILITY システム・ビュー」 783 ページ
- 「SYSCAPABILITYNAME システム・ビュー」 784 ページ

SYSCATALOG 統合ビュー

SYSCATALOG ビューの各ローには、システム・テーブルを記述します。

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"."SYSCATALOG"( creator,
  tname,dbspacename,tabletype,ncols,primary_key,"check",
  remarks)
  as select up.user_name,tab.table_name,file.dbspace_name,
    if tab.table_type = 'BASE' then 'TABLE' else tab.table_type
    endif,(select count(*) from SYS.SYSCOLUMN where
      SYSCOLUMN.table_id = tab.table_id),
    if tab.primary_root = 0 then 'N' else 'Y' endif,
    if tab.table_type <> 'VIEW' then tab.view_def endif,
    tab.remarks from
    SYS.SYSTABLE as tab key join
    SYS.SYSFILE as file join
    SYS.SYSUSERPERM as up on up.user_id = tab.creator
```

参照

- 「SYSTABLE 互換ビュー (旧式)」 858 ページ
- 「SYSFILE システム・ビュー」 792 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSCOLAUTH 統合ビュー

SYSCOLAUTH ビューの各ローには、カラムに付与されている権限セット (UPDATE、SELECT、または REFERENCES) を記述します。SYSCOLAUTH ビューは、「SYSCOLPERM システム・ビュー」 785 ページをユーザ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 up1.user_name,up2.user_name,up3.user_name,tab.table_name,
    col.column_name,cp.privilege_type,cp.is_grantable from
    SYS.SYSCOLPERM as cp join
    SYS.SYSUSERPERM as up1 on up1.user_id = cp.grantor join
    SYS.SYSUSERPERM as up2 on up2.user_id = cp.grantee join
    SYS.SYSTABLE as tab on tab.table_id = cp.table_id join
    SYS.SYSUSERPERM as up3 on up3.user_id = tab.creator join
    SYS.SYSCOLUMN 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」 785 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ
- 「SYSTABLE 互換ビュー (旧式)」 858 ページ

SYSCOLSTATS 統合ビュー

SYSCOLSTATS ビューには、オプティマイザ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.SYSCOLSTAT as s,SYS.SYSTABLE as t,SYS.SYSCOLUMN as c,SYS.SYSUSERPERM as u
where
    s.table_id = c.table_id and
    s.column_id = c.column_id and
    c.table_id = t.table_id and
    t.creator = u.user_id
```

参照

- 「SYSCOLSTAT システム・ビュー」 786 ページ
- 「SYSTABLE 互換ビュー (旧式)」 858 ページ
- 「SYSCOLUMN 互換ビュー (旧式)」 855 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSCOLUMNS 統合ビュー

SYSCOLUMNS ビューの各ローには、カタログ内の各テーブルとビューのカラムを記述します。

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"."SYSCOLUMNS"( creator,cname,tname,coltype,nulls,length,
  syslength,in_primary_key,colno,default_value,
  column_kind,remarks)
  as select up.user_name,col.column_name,tab.table_name,dom.domain_name,
```



```
col.nulls,col.width,col.scale,col.pkey,col.column_id,
col."default",col.column_type,col.remarks from
SYS.SYSCOLUMN as col join
SYS.SYSTABLE as tab on(tab.table_id = col.table_id) key join
SYS.SYSDOMAIN as dom join
SYS.SYSUSERPERM as up on up.user_id = tab.creator
```

参照

- 「SYSTABLE 互換ビュー (旧式)」 858 ページ
- 「SYSDOMAIN システム・ビュー」 789 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSFOREIGNKEYS 統合ビュー

SYSFOREIGNKEYS ビューの各ローには、カタログ内の各テーブルの外部キーを記述します。

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"."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,fk.role,
  (select list(string(fk_col.column_name,' IS ',
    pk_col.column_name)) from
    SYS.SYSFKCOL as fkc join SYS.SYSCOLUMN as fk_col on(
      fkc.foreign_table_id = fk_col.table_id
      and fkc.foreign_column_id = fk_col.column_id),

    SYS.SYSCOLUMN as pk_col where
    fkc.foreign_table_id = fk.foreign_table_id and
    fkc.foreign_key_id = fk.foreign_key_id and
    pk_col.table_id = fk.primary_table_id and
    pk_col.column_id = fkc.primary_column_id) from
  SYS.SYSFOREIGNKEY as fk join
  SYS.SYSTABLE as fk_tab on fk_tab.table_id = fk.foreign_table_id join
  SYS.SYSUSERPERM as fk_up on fk_up.user_id = fk_tab.creator join
  SYS.SYSTABLE as pk_tab on pk_tab.table_id = fk.primary_table_id join
  SYS.SYSUSERPERM as pk_up on pk_up.user_id = pk_tab.creator
```

参照

- 「SYSFKCOL 互換ビュー (旧式)」 855 ページ
- 「SYSCOLUMN 互換ビュー (旧式)」 855 ページ
- 「SYSFOREIGNKEY 互換ビュー (旧式)」 856 ページ
- 「SYSTABLE 互換ビュー (旧式)」 858 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 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、グループとメンバの多対多の関係を示します。グループには複数のメンバがあり、あるユーザは複数のグループのメンバになれます。

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.SYSGROUP,SYS.SYSUSERPERM as g,
    SYS.SYSUSERPERM as u where
    SYSGROUP.group_id = g.user_id
    and SYSGROUP.group_member = u.user_id
```

参照

- 「SYSGROUP システム・ビュー」 794 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 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 1 つのインデックスに関する記述です。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 up.user_name,idx.index_name,
    file.file_name,up.user_name,
    tab.table_name,
    if idx."unique" = 'N' then 'Non-unique' else
      if idx."unique" = 'U' then 'UNIQUE constraint'
      else 'Unique' endif
    endif,
    (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.SYSTABLE as tab key join
    SYS.SYSFILE as file key join
    SYS.SYSINDEX as idx join
    SYS.SYSUSERPERM as up on up.user_id = idx.creator
```

参照

- 「SYSINDEXES システム・ビュー」 798 ページ
- 「SYSTABCOL システム・ビュー」 826 ページ
- 「SYSFILE システム・ビュー」 792 ページ
- 「SYSINDEX 互換ビュー (旧式)」 856 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSOPTIONS 統合ビュー

SYSOPTIONS ビューの各ローには、SET コマンドを使用して作成されるオプションを記述します。各ユーザ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 ID PUBLIC に対する設定は、自分の設定を持たないユーザ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 up.user_name,opt."option",opt.setting from
    SYS.SYSOPTION as opt key join SYS.SYSUSERPERM as up
```

参照

- 「SYSOPTION システム・ビュー」 805 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSPROCAUTH 統合ビュー

SYSPROCAUTH ビューの各ローには、プロシージャに付与されている権限セットを記述します。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 up1.user_name,up2.user_name,p.proc_name from
    SYS.SYSPROCEDURE as p key join
    SYS.SYSPROCPERM as pp join
    SYS.SYSUSERPERM as up1 on up1.user_id = pp.grantee join
    SYS.SYSUSERPERM as up2 on up2.user_id = p.creator
```

参照

- 「SYSPROCEDURE システム・ビュー」 807 ページ
- 「SYSPROCPERM システム・ビュー」 810 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSPROCS 統合ビュー

SYSPROCS ビューには、プロシージャまたは関数に記録されているプロシージャ名、関数名、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,p.remarks from
    SYS.SYSPROCEDURE as p join
    SYS.ISYSUSER as u on u.user_id = p.creator
```

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に対するパラメータの記述です。

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"."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.SYSPROCEDURE as p on p.proc_id = pp.proc_id join
    SYS.ISYSUSER as up on up.user_id = p.creator join
    SYS.SYSDOMAIN as dom on dom.domain_id = pp.domain_id left outer join
    SYS.SYSUSERTYPE as ut on ut.type_id = pp.user_type
```

参照

- 「SYSPROCPARM システム・ビュー」 809 ページ
- 「SYSPROCEDURE システム・ビュー」 807 ページ
- 「SYSUSER システム・ビュー」 832 ページ
- 「SYSDOMAIN システム・ビュー」 789 ページ
- 「SYSUSERTYPE システム・ビュー」 834 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 パブリケーションに関する記述です。

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"."SYSPUBLICATIONS"
  as select u.user_name as creator,
    p.publication_name,
    p.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.SYSPUBLICATION as p join
    SYS.SYSUSERPERM as u on u.user_id = p.creator
```

参照

- 「SYSPUBLICATION システム・ビュー」 811 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 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」 813 ページ
- 「SYSREMOTEOPTIONTYPE システム・ビュー」 814 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・パラメータの値に関する記述です。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」 814 ページ
- 「SYSREMOTEOPTIONTYPE システム・ビュー」 814 ページ
- 「SYSREMOTEOPTION システム・ビュー」 813 ページ

- 「SYSUSER システム・ビュー」 832 ページ

SYSREMOTETYPES 統合ビュー

SQL Remote のメッセージ・タイプについて 1 つずつ、パブリッシャ・アドレスを含めて、SYSREMOTETYPES ビューの各ローで示します。

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"."SYSREMOTETYPES"
  as select SYSREMOTETYPE.type_id,SYSREMOTETYPE.type_name,
     SYSREMOTETYPE.publisher_address,SYSREMOTETYPE.remarks
     from SYS.SYSREMOTETYPE
```

参照

- 「SYSREMOTETYPE システム・ビュー」 814 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 ID (サブスクライバ) を記述します。そのユーザに送信された SQL Remote メッセージと、そのユーザ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.SYSREMOTEUSER 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.SYSREMOTEUSER as r key join
   SYS.SYSUSERPERM as u key join
   SYS.SYSREMOTETYPE as t
```

参照

- 「SYSREMOTEUSER システム・ビュー」 815 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ
- 「SYSREMOTETYPE システム・ビュー」 814 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 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.SYSSUBSCRIPTION as s key join
    SYS.SYSPUBLICATION as p join
    SYS.SYSUSERPERM as u on u.user_id = s.user_id
```

参照

- 「SYSSUBSCRIPTION システム・ビュー」 820 ページ
- 「SYSPUBLICATION システム・ビュー」 811 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 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 from
    SYS.ISYSSYNC
```

参照

- 「SYSSYNC システム・ビュー」 821 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 key join SYS.ISYSPUBLICATION
     as p where s.site_name is null
```

参照

- 「SYSSYNC システム・ビュー」 821 ページ
- 「SYSPUBLICATION システム・ビュー」 811 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」 821 ページ
- 「SYSPUBLICATION システム・ビュー」 811 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、未加工のデータとは異なり、ユーザ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

- 「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』
- 「SYSPUBLICATION システム・ビュー」 811 ページ
- 「SYSPROCEDURE システム・ビュー」 807 ページ
- 「SYSSYNCSCRIPT システム・ビュー」 822 ページ

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 key join SYS.ISYSPUBLICATION
     as p 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」 821 ページ
- 「SYSPUBLICATION システム・ビュー」 811 ページ

- 「SYSSYNCUSERS 統合ビュー」 851 ページ

SYSSYNCUSERS 統合ビュー

Mobile Link 同期ユーザに対応する同期設定のビュー。

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」 821 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 up1.user_name,up2.user_name,up3.user_name,tab1.table_name,
    up4.user_name,tab2.table_name,tp.selectauth,tp.insertauth,
    tp.deleteauth,tp.updateauth,tp.updatecols,tp.alterauth,
    tp.referenceauth from
    SYS.SYSTABLEPERM as tp join
    SYS.SYSUSERPERM as up1 on up1.user_id = tp.grantor join
    SYS.SYSUSERPERM as up2 on up2.user_id = tp.grantee join
    SYS.SYSTABLE as tab1 on tab1.table_id = tp.stable_id join
    SYS.SYSUSERPERM as up3 on up3.user_id = tab1.creator join
    SYS.SYSTABLE as tab2 on tab2.table_id = tp.ttable_id join
    SYS.SYSUSERPERM as up4 on up4.user_id = tab2.creator
```

参照

- 「SYSTABLEPERM システム・ビュー」 828 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ
- 「SYSTABLE 互換ビュー (旧式)」 858 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に関する記述です。このテーブルには、参照トリガ・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 up.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 key join
    SYS.ISYSTAB as tab join
    SYS.SYSUSERPERM as up on up.user_id = tab.creator where
    trig.foreign_table_id is null
```

参照

- 「SYSTRIGGER システム・ビュー」 830 ページ
- 「SYSTAB システム・ビュー」 823 ページ
- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSUSEROPTIONS 統合ビュー

SYSUSEROPTIONS ビューには、各ユーザに有効なオプション設定が含まれます。ユーザ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.name,
    o."option",
    isnull((select s.setting from
```

```
  SYS.SYSOPTIONS as s where
  s.user_name = u.name and
  s."option" = o."option"),
o.setting) from
SYS.SYSOPTIONS as o,SYS.SYSUSERAUTH as u where
o.user_name = 'PUBLIC'
```

参照

- 「SYSOPTIONS 統合ビュー」 844 ページ
- 「SYSUSERAUTH 互換ビュー (旧式)」 859 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ています。

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,t.view_def from
    SYS.SYSTABLE as t join
    SYS.ISYSUSER as u on(u.user_id = t.creator) where
    t.table_type = 'VIEW'
```

カラム名	カラム型	カラム制約
vcreator	CHAR(128)	NOT NULL
viewname	CHAR(128)	NOT NULL
viewtext	LONG VARCHAR	

参照

- 「SYSTABLE 互換ビュー (旧式)」 858 ページ
- 「SYSUSER システム・ビュー」 832 ページ

互換ビュー

互換ビューは、SQL Anywhere の 10.0.0 バージョンと互換性があるビューです。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・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』
- 「DB_PROPERTY 関数 [システム]」 148 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」 144 ページ

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・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』
- 「DB_PROPERTY 関数 [システム]」 148 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」 144 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 SQL Anywhere 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」 826 ページに対応する 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.ISYSTABCOL as b
    left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
    left outer join SYS.SYSIDXCOL 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」 826 ページ
- 「SYSREMARK システム・ビュー」 812 ページ
- 「SYSINDEXES システム・ビュー」 798 ページ

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・カラムの関係を記述します。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.SYSIDXCOL as a,
    SYS.SYSIDX as b
    where a.table_id = b.table_id
    and a.index_id = b.index_id
    and b.index_category = 2
```


参照

- 「SYSIDX システム・ビュー」 796 ページ
- 「SYSINDEXES システム・ビュー」 798 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 SQL Anywhere 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」 793 ページに対応する 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.SYSIDX 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」 796 ページ
- 「SYSPHYSIDX システム・ビュー」 806 ページ
- 「SYSREMARK システム・ビュー」 812 ページ
- 「SYSFKEY システム・ビュー」 793 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 SQL Anywhere 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」 796 ページに対応する 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.file_id,
    case b."unique"
    when 1 then 'Y'
    when 2 then 'U'
    when 3 then 'M'
    when 4 then 'N'
    else 'I'
    end as "unique",
    t.creator,
    b.index_name,
    r.remarks,
    10 as hash_limit
    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.SYSTABLE as t
    where t.table_id = b.table_id
    and b.index_category = 3
```

参照

- 「SYSIDX システム・ビュー」 796 ページ
- 「SYSPHYSIDX システム・ビュー」 806 ページ
- 「SYSTABLE 互換ビュー (旧式)」 858 ページ
- 「SYSREMARK システム・ビュー」 812 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・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』
- 「DB_PROPERTY 関数 [システム]」 148 ページ
- 「DB_EXTENDED_PROPERTY 関数 [システム]」 144 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 SQL Anywhere 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。「SYSINDEXES システム・ビュー」 798 ページの使用をおすすめします。

SYSIXCOL の各ローには、インデックスのカラムを記述します。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"."SYSIXCOL"
  as select a.table_id,
    a.index_id,
    a.sequence,
    a.column_id,
    a."order"
    from SYS.SYSIDXCOL as a,
     SYS.SYSIDX 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」 796 ページ
- 「SYSINDEXES システム・ビュー」 798 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 SQL Anywhere 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」 823 ページに対応する 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、データベースの 1 つのテーブルを記述します。

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"."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,
case b.table_type
when 1 then 'BASE'
when 2 then 'MAT VIEW'
when 3 then 'GBL TEMP'
when 4 then 'LCL TEMP'
when 21 then 'VIEW'
when 22 then 'JVT'
else 'INVALID'
end 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.ISYSTAB 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」 823 ページ
- 「SYSREMARK システム・ビュー」 812 ページ
- 「SYSSOURCE システム・ビュー」 819 ページ
- 「SYSVIEW システム・ビュー」 835 ページ
- 「SYSPROXYTAB システム・ビュー」 811 ページ
- 「SYSIDX システム・ビュー」 796 ページ
- 「SYSPHYSIDX システム・ビュー」 806 ページ

SYSUSERAUTH 互換ビュー (旧式)

SYSUSERAUTH ビューは、古いバージョンの SQL Anywhere 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。「SYSUSERAUTHORITY システム・ビュー」 833 ページを参照してください。

SYSUSERAUTH ビューの各ローには、user_id を公開せずにユーザを記述します。代わりに、各ユーザは、ユーザ名で識別されます。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,SYSUSERPERM.dbaauth,
     SYSUSERPERM.scheduleauth,SYSUSERPERM.user_group
    from SYS.SYSUSERPERM
```

参照

- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ

SYSUSERLIST 互換ビュー (旧式)

SYSUSERAUTH ビューは、古いバージョンの SQL Anywhere との互換性を保つために用意されています。

SYSUSERLIST ビューの各ローには、user_id とパスワードを公開することなく、ユーザに関して記述します。各ユーザは、ユーザ名で識別されます。

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,SYSUSERPERM.scheduleauth,
     SYSUSERPERM.user_group
    from SYS.SYSUSERPERM
```

参照

- 「SYSUSERPERM 互換ビュー (旧式)」 860 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 1 人のユーザ ID を記述します。

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"."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」 833 ページ
- 「SYSUSER システム・ビュー」 832 ページ
- 「SYSREMARK システム・ビュー」 812 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 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 ビューの各ローには ユーザ ID が記述されます。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, SYSUSERPERM.dbaauth,
  SYSUSERPERM.scheduleauth, SYSUSERPERM.user_group,
  SYSUSERPERM.publishauth, SYSUSERPERM.remotedbaauth,
  SYSUSERPERM.remarks
  from SYS.SYSUSERPERM
```

参照

- 「SYSUSERPERM 互換ビュー (旧式)」 860 ページ
- 「SYSUSERAUTHORITY システム・ビュー」 833 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、**dbo** という特殊なユーザが所有しています。一部はマスタ・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 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、またプロシージャのパラメータごとに 1 ロー
syscomments	ビュー、ルール、デフォルト、トリガ、プロシージャごとに 1 つ以上のロー。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、ルール、トリガ、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
systypes	システム提供またはユーザ定義のデータ型ごとに 1 ロー
sysusers	データベースについての許可を持つユーザごとに 1 ロー
syslogins	有効なユーザ・アカウントごとに 1 ロー

第 7 章

システム・プロシージャ

目次

システム・プロシージャの概要

SQL Anywhere に含まれるシステム・プロシージャ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、カタログ・システム・プロシージャ。
- 電子メールのサポートとその他の関数用の拡張システム・プロシージャ。
- サーバの動作を制御するためのその他のプロシージャ。
- Transact-SQL システムとカタログ・プロシージャ。「Adaptive Server Enterprise のシステム・プロシージャとカタログ・プロシージャ」 998 ページを参照してください。

「システム関数」 100 ページを参照してください。

システム・プロシージャの定義

システム・プロシージャと関数の詳細については、Sybase Central を参照してください。

- システム・プロシージャと関数を表示するには、接続しているデータベースを右クリックし、[所有者別にオブジェクトをフィルタ] を選択してから [dbo] を選択します。
- データベースの [プロシージャとファンクション] フォルダを開きます。
- プロシージャ定義を表示するには、左ウィンドウ枠でプロシージャを選択してから、右ウィンドウ枠で [SQL] タブをクリックします。

システム・プロシージャ

システム・プロシージャは、dbo ユーザによって所有されています。これらのプロシージャのなかには、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以外について説明します。Windows CE 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

XML ドキュメントから結果セットを生成します。

構文

openxml(*xml_data*,
xpath **[,** *flags* **[,** *namespaces* **]])**
WITH (*column-name column-type* [*xpath*],... **)**

引数

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。

xpath　XPath クエリを含む文字列。XPath を使用すると、クエリ対象の 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、属性として 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。クエリに対するスコープ内のネームスペースは、文書のルート要素から取得されます。ネームスペースを指定しない場合、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。

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。

openxml の WITH 句構文は、ストアド・プロシージャから選択する際に使用する構文と似ています。

ストアド・プロシージャからの選択については、「FROM 句」 552 ページを参照してください。

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。「SQL データ型」 47 ページを参照してください。

使用法

openxml システム・プロシージャは、*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・プロシージャで使用される XML パーサ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 エンコード形式とみなされ、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・プロシージャを使用して復号化できます。「FOR XML とバイナリ・データ」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

openxml システム・プロシージャは、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' と同じで、'..' 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' などです。
- 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 を持つノード a を検索し、a [b="i"] は、i のテキスト値を持つ少なくとも 1 つの子 b を持つノード a を検索します。

参照

- 「XPath 式の使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「openxml を使用した XML のインポート」 『SQL Anywhere サーバ - SQL の使用法』
- 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、ドット (.) を使用します。

```
SELECT * FROM openxml( '<products>
            <ProductType ID="301">Tee &amp;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')
```

openxml システム・プロシージャの使用例については、「openxml を使用した XML のインポート」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

sa_ansi_standard_packages システム・プロシージャ

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(*sql-standard-string*, *sql-statement-string* **)**

引数

- ***sql-standard-string*** コア拡張機能に使用する規格。SQL:1999 または SQL:2003。
- ***sql-statement-string*** 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」『SQL Anywhere サーバ - プログラミング』
- 「SQLFLAGGER 関数 [その他]」 257 ページ
- 「sql_flagger_error_level オプション [互換性]」『SQL Anywhere サーバ - データベース管理』
- 「sql_flagger_warning_level オプション [互換性]」『SQL Anywhere サーバ - 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

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

監査が有効になると、システム・プロシージャではコメントが監査ログに追加されます。文字列は最大の長さが 200 バイトです。

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

- 「auditing オプション [データベース]」『SQL Anywhere サーバ - 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 - 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監査ログに追加します。

```
CALL sa_audit_string( 'Auditing test' )
```

sa_check_commit システム・プロシージャ

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
- ***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

備考

データベース・オプション wait_for_commit が ON に設定されているか、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た場合、参照整合性に違反する方法でデータベースを更新できます。ただし、変更がコミットされる前にこれらの違反を解決する必要があります。

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

DBA 権限が必要です。

関連する動作

なし。

参照

- 「wait_for_commit オプション [データベース]」『SQL Anywhere サーバ - データベース管理』
- 「CREATE TABLE 文」 460 ページ

例

次の一連のコマンドを Interactive SQL から実行できます。これらの一連のコマンドは、サンプル・データベースの Departments テーブルから、参照整合性に違反する方法でローを削除します。sa_check_commit システム・プロシージャを呼び出して、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

データベース・クリーナを起動し、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・プロシージャを使用すると、データベース・クリーナを強制的にただちに実行したり、クリーナの起動のたびにその実行時間を指定したりできます。

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。

データベース・クリーナは、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・クリーナは、データベース・サーバ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・サーバの現在のアクティビティ量、データベース・クリーナがすでにクリーンニングに費やした時間など、数多くの要因に左右されます。0.5 秒間実行した後に、クリーナがサーバでアクティブな要求を検出した場合、クリーナは停止し、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・クリーナは、サーバ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、サーバの休止時間を活用します。

データベース・クリーナの統計情報は、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。
- **CleanabledRowsCleaned**　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・プロシージャを使用すると、データベース・クリーナ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・クリーナを起動するイベントを設定します。「CREATE EVENT 文」 397 ページを参照してください。

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 EVENT 文」 397 ページ
- CleanablePagesAdded、CleanablePagesCleaned、CleanableRowsAdded、CleanableRowsCleaned プロパティ：「データベース・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』

例

次の例では、データベース・クリーナの実行時間を 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 20% 以上のページ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.file_id ) -
                DB_EXTENDED_PROPERTY( 'FreePages', t.file_id ) ))
    INTO @num_db_pages
    FROM (SELECT file_id FROM SYSFILE) 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

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、オプティマイザが使用するために保守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*** リモート・テーブルの所有者を指定するオプションの CHAR(128) パラメータ。このパラメータ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*** このオプションの CHAR(128)パラメータは、統計情報を計算するカラムを指定します。このパラメータ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*** リモート・テーブルの所有者を指定するオプションの CHAR(128) パラメータ。このパラメータが指定されていない場合、指定した *tab_name* と一致する最初のテーブルの所有者をデータベース・サーバが使用します。
- ***max_rows*** オプションの INTEGER パラメータは、計算に使用する行数を指定します。このパラメータを指定しない場合、デフォルト値は 1000 行です。0 を指定すると、データベース・サーバはテーブル内のすべての行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 文字列長の標準偏差平均。
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 文字列長の標準偏差平均。
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・サーバはデータの先頭 1000 ローのみを使用します。

avg_compression_ratio の場合、値は 100 以下にすることはできません。ただし、高度に圧縮可能なデータが圧縮カラムに挿入された場合 (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場合の 80 % 未満です。

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

- 「カラムを圧縮するかどうかの選択」 『SQL Anywhere サーバ - SQL の使用法』

例

次の例では、SELECT 文で sa_column_stats システム・プロシージャを使用して、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。

```
SELECT * FROM sa_column_stats()
  WHERE num_values_compressed > 0
  ORDER BY avg_compression_ratio desc;
```

次の例では、前の例からの選択を絞り込み、bsmith が所有されてい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

サーバ上の指定したデータベース接続に関して、最後に作成された SQL 文を返します。

構文

sa_conn_activity([*connidparm*] **)**

引数

- ***connidparm*** オプションの INTEGER パラメータを使用して、接続の ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INT	接続の ID 番号。
Name	VARCHAR(255)	接続の名前。
Userid	VARCHAR(255)	接続のユーザ ID。
DBNumber	INT	データベースの ID 番号。
LastReqTime	VARCHAR(255)	指定の接続に対する最後の要求が開始した時間。
LastStatement	LONG VARCHAR	接続で最後に作成された SQL 文。

備考

サーバ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');
```

このプロシージャは、データベース・サーバ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、要求ロギングの代わりとして使用できます。

これらの値の導出元の LastStatement プロパティの詳細については、「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

connidparm を指定しない場合、データベース・サーバ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* がゼロ未満の場合、現在の接続のオプション値が返されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』
- 「-zl サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』
- 「sa_server_option システム・プロシージャ」 948 ページ

sa_conn_compression_info システム・プロシージャ

通信の圧縮率を要約します。

構文

sa_conn_compression_info([*connidparm*] **)**

引数

- ***connidparm*** オプションの INTEGER パラメータを使用して、接続の ID 番号を指定します。

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、サーバへの全接続 (Server) の圧縮統計かを識別する文字列。

カラム名	データ型	説明
ConnNumber	INTEGER	接続 ID を示す INTEGER 。Type が Server の場合は、NULL が返されます。
Compression	VARCHAR(10)	その接続に対して圧縮が有効になっているかどうかを示す文字列。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。
TotalBytesUnComp	INTEGER	圧縮が無効な場合の送受信バイト数を示す INTEGER。
CompRate	NUMERIC(5.2)	全体的な圧縮率を示す NUMERIC (5,2)。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、データの圧縮率が 75% であったということになり、元のサイズの 3/4 に圧縮されたことを意味します。
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)。
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)。
TotalPackets	INTEGER	送受信された実際のパケット数の合計を示す INTEGER。
TotalPacketsUnComp	INTEGER	圧縮が無効な場合の送受信パケット数の合計を示す INTEGER。
CompPktRate	NUMERIC(5.2)	パケットの全体的な圧縮率を示す NUMERIC (5,2)。
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)。
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)。

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。*connection-id* を指定しない場合は、このシステム・プロシージャによって、サーバ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。

これらの値の導出元プロパティの詳細については、「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

パーミッション

なし。

関連する動作

なし。

例

次の例では、sa_conn_compression_info システム・プロシージャを使用して、サーバ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

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*** オプションの INTEGER パラメータを使用して、接続の ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号。
Name	VARCHAR (255)	接続の名前。
Userid	VARCHAR (255)	接続のユーザ ID。
DBNumber	INTEGER	データベースの ID 番号。
LastReqTime	VARCHAR (255)	指定の接続に対する最後の要求が開始した時間。
ReqType	VARCHAR (255)	最後の要求のタイプを示す文字列。

カラム名	データ型	説明
CommLink	VARCHAR (255)	接続用の通信リンク。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、同一コンピュータ接続の場合は local となります。
NodeAddr	VARCHAR (255)	クライアント／サーバ接続のクライアント・アドレス。
ClientPort	INTEGER	TCP/IP を使用してクライアント・アプリケーションが通信するポート番号。
ServerPort	INTEGER	TCP/IP を使用してサーバが通信するポート番号。
BlockedOn	INTEGER	現在の接続が制限されていない場合は 0。ブロックされている場合は、ロック矛盾によってブロックされる接続の数。
LockTable	VARCHAR (255)	接続がロックを待機している場合、LockTable の名前は、待機しているロックに関連付けられているテーブルの名前になります。それ以外の場合は、LockTable は空の文字列です。
UncommitOps	INTEGER	コミットされていないオペレーションの数。
LockRowID	UNSIGNED BIGINT	特定のロー識別子に関連付けられているロックで接続が待機している場合は、LockRowID にそのロー識別子が含まれます。ローに関連付けられているロックで接続が待機していない場合 (つまり、ロックで待機していない場合、または関連付けられているローがないロックで待機している場合)、LockRowID は NULL です。
LockIndexID	INTEGER	特定のインデックスに関連付けられているロックで接続が待機している場合は、LockIndexID にそのインデックスの識別子が含まれます (ロックが、LockTable のテーブルの全インデックスに関連付けられている場合は、-1 が含まれます)。インデックスに関連付けられているロックで接続が待機していない場合 (つまり、ロックで待機していない場合、または関連付けられているインデックスがないロックで待機している場合)、LockIndexID は NULL です。

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・プロシージャによって、サーバ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* がゼロ未満の場合、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、このプロシージャが返す BlockedOn 値によって、どのユーザがどのユーザ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することができます。

パーミッション

なし。

関連する動作

なし。

参照

- 「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』
- 「sa_locks システム・プロシージャ」 915 ページ
- 「SYSIDX システム・ビュー」 796 ページ

例

次の例では、sa_conn_info システム・プロシージャを使用して、サーバ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	...
...	...	...	...	...

sa_conn_list システム・プロシージャ

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*** オプションの INTEGER パラメータを使用して、接続の ID 番号を指定します。
- ***dbidparm*** オプションの INTEGER パラメータを使用して、データベースの ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号。

備考

パラメータを指定しない場合、または両方のパラメータが NULL の場合、データベース・サーバ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* がゼロ未満の場合、現在の接続の接続 ID 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。

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」 888 ページ
- 「sa_conn_options システム・プロシージャ」 883 ページ

sa_conn_options システム・プロシージャ

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*** オプションの INTEGER パラメータを使用して、接続の ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号。
PropNum	INTEGER	接続プロパティの番号。
OptionName	VARCHAR(255)	オプションの名前。
OptionDescription	VARCHAR(255)	オプションの説明。
Value	LONG VARCHAR	オプションの値。

備考

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 Number、OptionName、OptionDescription、Value の接続 ID 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* がゼロ未満の場合、現在の接続のオプション値が返されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_db_list システム・プロシージャ」 888 ページ
- 「sa_conn_list システム・プロシージャ」 882 ページ

sa_conn_properties システム・プロシージャ

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*** オプションの INTEGER パラメータを使用して、接続の ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号。
PropNum	INTEGER	接続プロパティの番号。

カラム名	データ型	説明
PropName	VARCHAR(255)	接続プロパティの名前。
PropDescription	VARCHAR(255)	接続プロパティの説明。
Value	LONG VARCHAR	接続プロパティの値。

備考

使用可能な各接続プロパティについて Number、PropNum、PropName、PropDescription、Value の接続 ID 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。

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* がゼロ未満の場合、現在の接続のオプション値が返されます。

パーミッション

なし。

関連する動作

なし。

参照

- 「sa_conn_list システム・プロシージャ」 882 ページ
- 「sa_conn_options システム・プロシージャ」 883 ページ
- 「システム関数」 100 ページ
- 「接続レベルのプロパティ」 『SQL Anywhere サーバ - 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 氏です。

sa_convert_ml_progress_to_timestamp システム・プロシージャ

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捗値を 64 ビット INTEGER 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 である 1 つのパラメータを使用します。

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」 886 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( 3600000 );
```

sa_convert_timestamp_to_ml_progress システム・プロシージャ

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捗値を TIMESTAMP から 64 ビット UNISIGNED BIGINT に変換します。

構文

sa_convert_timestamp_to_ml_progress([*t1*] **)**

引数

- ***t1*** オプションの TIMESTAMP パラメータを使用して、進捗値を 64 ビット UNISIGNED BIGINT に変換するように指定します。

備考

この関数は、パラメータとして渡されるタイムスタンプを表す UNISIGNED BIGINT を返します。このプロシージャは、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」 886 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_db_info システム・プロシージャ

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*** オプションの INTEGER パラメータを使用して、データベースの ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INTEGER	接続の ID 番号。
Alias	VARCHAR(255)	データベース名。
File	VARCHAR(255)	データベース・ルート・ファイルのパスを含む名前。
ConnCount	INTEGER	データベースとの接続の数。
PageSize	INTEGER	データベースのページ・サイズ (バイト)。
LogName	VARCHAR(255)	トランザクション・ログのパスを含むファイル名。

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。

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」 889 ページ
- 「データベース・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』

例

次の文は、サーバで実行される各データベースのローを返します。

```
CALL sa_db_info( );
```

プロパティ	Value
Number	0
Alias	demo
File	C:¥Documents and Settings¥All Users ¥Documents¥SQL Anywhere 10¥Samples ¥demo.db
ConnCount	1
PageSize	4096
LogName	C:¥Documents and Settings¥All Users ¥Documents¥SQL Anywhere 10¥Samples ¥demo.log

sa_db_list システム・プロシージャ

データベース ID を返します。

構文

sa_db_list([*dbidparm*] **)**

引数

- ***dbidparm*** オプションの INTEGER パラメータを使用して、データベースの ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INTEGER	データベースの 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。

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」 882 ページ
- 「sa_conn_options システム・プロシージャ」 883 ページ

sa_db_properties システム・プロシージャ

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*** オプションの INTEGER パラメータを使用して、データベースの ID 番号を指定します。

結果セット

カラム名	データ型	説明
Number	INTEGER	データベースの 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。

カラム名	データ型	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・プロシージャは使用可能な各データベース・プロパティについてデータベース ID 番号と、PropNum、PropName、PropDescription、Value を返します。すべてのデータベース・プロパティと、データベースに関する統計情報の値が返されます。

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。

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」 887 ページ
- 「データベース・レベルのプロパティ」 『SQL Anywhere サーバ - 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	...
0	1	IdleCheck	...
0	2	IdleWrite	...
...	...	...	...

sa_dependent_views システム・プロシージャ

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*** この CHARACTER パラメータを使用して、テーブルまたはビューの名前を指定します。
- ***owner_name*** オプションの CHARACTER パラメータを使用して、*tbl_name* の所有者を指定します。

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のプロシージャを使用できます。

パーミッション

DBA 権限が必要です。

関連する動作

なし。

参照

- 「SYSDEPENDENCY システム・ビュー」 788 ページ
- 「ビューの依存性」 『SQL Anywhere サーバ - SQL の使用法』

例

次の例では、sa_dependent_views システム プロシージャを使用して、SalesOrders テーブルに依存するビューの ID リストを取得します。このプロシージャは、SalesOrders の場合に table_id、従属ビュー ViewSalesOrders の場合には dep_view_id を返します。

```
sa_dependent_views( 'SalesOrders' );
```

次の例では、sa_dependent_views システム・プロシージャを 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_query システム・プロシージャ

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*** この LONG VARCHAR パラメータを使用して、記述されている SQL 文のテキストを指定します。
- ***add_keys*** オプションの BIT パラメータを使用して、記述されているクエリの結果セットで一意にローを識別するカラムのセットを決定するかどうかを指定します。デフォルトは 0 です。データベース・サーバはカラムの識別を試行しません。このパラメータの詳細については、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序数位置 (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。「SYSDOMAIN システム・ビュー」 789 ページを参照してください。
domain_name	VARCHAR(128)	データ型の名前。「SYSDOMAIN システム・ビュー」 789 ページを参照してください。
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	文字列パラメータでは長さ、数値パラメータでは精度、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	文字列パラメータでは長さ、数値パラメータでは精度、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 つの場合はユーザ定義のデータ型の type_id、それ以外の場合は NULL。「SYSUSERTYPE システム・ビュー」 834 ページを参照してください。

カラム名	データ型	説明
user_type_name	VARCHAR(128)	1 つの場合はユーザ定義のデータ型の name、それ以外の場合は NULL。「SYSUSERTYPE システム・ビュー」 834 ページを参照してください。
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	式がフィールドの場合は table_id、それ以外の場合は NULL。「SYSTAB システム・ビュー」 823 ページを参照してください。
base_column_id	UNSIGNED INTEGER	式がフィールドの場合は column_id、それ以外の場合は NULL。「SYSTABCOL システム・ビュー」 826 ページを参照してください。
base_owner_name	VARCHAR(128)	式がフィールドの場合は所有者名、それ以外の場合は NULL。「SYSUSER システム・ビュー」 832 ページを参照してください。
base_table_name	VARCHAR(128)	式がフィールドの場合はテーブル名、それ以外の場合は NULL。
base_column_name	VARCHAR(128)	式がフィールドの場合はカラム名、それ以外の場合は NULL。
nulls_allowed	BIT	式を NULL にできる場合 1、それ以外の場合は 0 を示すインジケータ。
is_autoincrement	BIT	式が自動増分すると宣言されているカラムの場合 1、それ以外の場合は 0 を示すインジケータ。
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。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。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関する名前と型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一意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

ます。オプティマイザがキーを検索する場合、カラムまたはキーで使用されるカラムは、値が 1 の is_key_column によって結果のキーが識別されます。キーが見つからない場合、エラーが返されます。

キーに含まれるがクエリで明示的に参照されないカラムの場合、is_added_key_column は 1 に設定されます。これは、カラムがプロシージャの結果に追加されたことを示します。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、オプティマイザは結果セットのキーを検索しようとしません。また、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るカラムのサイズを記述します。クライアントにフェッチするときに、width 値は、フィールドのサイズを指定します。クライアントの型表記は、データベース・サーバ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フィールドは、CREATE TABLE と一致する declared_width 値があります。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]」 165 ページ
- 「文字データ型」 48 ページ
- 「return_date_time_as_string オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』

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' );
```

この結果、*add_keys* パラメータが指定されなかったため、is_key_column と is_added_key_column の値を NULL と示します。

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 が示されます。これは、クエリの結果セットに含まれるローを一意に識別するときに必要なカラムが Employees.EmployeeID and 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、記述されているクエリで明示的にされなかったためです。

sa_disable_auditing_type システム・プロシージャ

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*** この VARCHAR(128) パラメータを使用して、カンマで区切られた文字列を指定します。文字列には、次の値のいずれかまたは複数が含まれます。

  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・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・チェックの監査を無効にします。

  triggers トリガ・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 権限が必要です。

関連する動作

なし。

参照

- 「データベース・アクティビティの監査」 『SQL Anywhere サーバ - データベース管理』
- 「auditing オプション [データベース]」 『SQL Anywhere サーバ - 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

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*** この VARCHAR(128) パラメータを使用して、DB 領域、ログ・ファイル、ミラー・ログ・ファイル、またはテンポラリ・ファイルの名前を指定します。

  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。

  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。「事前定義の DB 領域」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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、ミラー・ログ・ファイル、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。

備考

p_dbspace_name パラメータ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、トランザクション・ログ・ミラー、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)。

SQL Anywhere データベースの事前定義の DB 領域の名前リストについては、「事前定義の DB 領域」 『SQL Anywhere サーバ - データベース管理』を参照してください。

パーミッション

DBA 権限が必要です。

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
SYSTEM	10952101888
Transaction Log	10952101888
Temporary File	10952101888

sa_enable_auditing_type システム・プロシージャ

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*** この VARCHAR(128) パラメータを使用して、カンマで区切られた文字列を指定します。文字列には、次の値のいずれかまたは複数が含まれます。

  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・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・チェックの監査を有効にします。

triggers　トリガ・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 権限が必要です。

関連する動作

なし。

参照

- 「データベース・アクティビティの監査」『SQL Anywhere サーバ - データベース管理』
- 「auditing オプション [データベース]」『SQL Anywhere サーバ - 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

データベース・サーバ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・プロパティの番号。
PropName	VARCHAR(255)	データベース・サーバ・プロパティ名。
PropDescription	VARCHAR(255)	データベース・サーバ・プロパティの説明。
Value	LONG VARCHAR	データベース・サーバ・プロパティの値。

備考

使用可能な各サーバ・プロパティの PropNum、PropName、PropDescription、Value を返します。すべてのデータベース・サーバ・プロパティと、データベース・サーバに関する統計情報の値が返されます。使用可能なデータベース・サーバ・プロパティのリストについては、「システム関数」 100 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「サーバ・レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』

例

次の文は、使用可能な一連のサーバ・プロパティを返します。

```
CALL sa_eng_properties( );
```

PropNum	PropName	…
1	IdleWrite	…
2	IdleChkPt	…
…	…	…

sa_flush_cache システム・プロシージャ

データベース・サーバ・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、このプロシージャを使用して現在のデータベースのデータベース・サーバ・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 権限が必要です。

関連する動作

なし。

sa_flush_statistics システム・プロシージャ

すべてのコスト・モデル統計をデータベース・サーバ・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、このプロシージャを実行する必要はありません。

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

- 「sa_get_histogram システム・プロシージャ」 903 ページ
- 「SYSCOLSTAT システム・ビュー」 786 ページ
- 「ヒストグラム・ユーティリティ (dbhist)」 『SQL Anywhere サーバ - データベース管理』

sa_get_bits システム・プロシージャ

ビット文字列を取得し、文字列で各ビットのローを返します。デフォルトで、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*** この LONG VARBIT パラメータを使用して、ビットを取得するビット文字列を指定します。*bit_string* パラメータが NULL の場合、ローは返されません。
- ***only_on_bits*** オプションの BIT を使用して、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します。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、1 の bitnum です。
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。

備考

sa_get_bits システム・プロシージャは、ビット文字列を復号化し、ビットの値を示すビット文字列の各ビットについて 1 つのローを返します。*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。最適化によって、オンのビットがいくつかある長いビット文字列の場合に、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 にオンのビットを示す、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、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」 962 ページ
- 「SET_BIT 関数 [ビット配列]」 245 ページ
- 「SET_BITS 関数 [集合]」 246 ページ

- 「GET_BIT 関数 [ビット配列]」 168 ページ

例

次の例は、sa_get_bits システム・プロシージャを使用して、整数のセットを文字列としてコード化してから、ジョインで使用するとき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

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*** UNSIGNED SMALLINT パラメータを使用して、データベース・ファイル ID を指定します。

備考

file_id はシステム・テーブル SYSFILE から取得できます。

このプロシージャは、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

DBA 権限が必要です。

関連する動作

なし。

参照

- 「ISYSFILE システム・テーブル」 754 ページ
- 「SYSOPTSTAT システム・ビュー」 805 ページ

sa_get_histogram システム・プロシージャ

カラムのヒストグラムを取り出します。

構文

```
sa_get_histogram(
  col_name,
  tbl_name
  [, owner_name ]
)
```

引数

- ***col_name*** CHAR(128) パラメータを使用して、ヒストグラムを取得するカラムを指定します。
- ***tbl_name*** CHAR(128) パラメータを使用して、*col_name* が検出されたテーブルを指定します。
- ***owner_name*** オプションの CHAR(128) パラメータを使用して、*tbl_name* の所有者を指定します。

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・サーバからカラム統計を取り出します。この統計情報をシステム・テーブル ISYSCOLSTAT に永続的に格納されている場合、サーバが実行中にメモリに保守され、定期的に ISYSCOLSTAT に書き込みます。この結果、sa_get_histogram システム・プロシージャが返す統計情報は、任意の時点で ISYSCOLSTAT から選択して取得した情報とは異なります。

sa_flush_statistics システム・プロシージャ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。

ヒストグラムの表示には、ヒストグラム・ユーティリティを使用することをおすすめします。「ヒストグラム・ユーティリティ (dbhist)」『SQL Anywhere サーバ - データベース管理』を参照してください。

文字列カラムに対する述部の選択性を決定するには、ESTIMATE または ESTIMATE_SOURCE 関数を使用してください。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。

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

- 「オプティマイザの推定とカラム統計」『SQL Anywhere サーバ - SQL の使用法』
- 「ヒストグラム・ユーティリティ (dbhist)」『SQL Anywhere サーバ - データベース管理』
- 「ESTIMATE 関数 [その他]」 157 ページ
- 「ESTIMATE_SOURCE 関数 [その他]」 157 ページ
- 「ISYSCOLSTAT システム・テーブル」 753 ページ
- 「sa_flush_statistics システム・プロシージャ」 900 ページ

sa_get_request_profile システム・プロシージャ

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*** オプションの LONG VARCHAR パラメータを使用して、要求ロギングのファイル名を指定します。
- ***conn_id*** オプションの UNSIGNED INTEGER パラメータを使用して、接続の ID 番号を指定します。

- ***first_file*** オプションの INTEGER パラメータを使用して、分析する最初の要求ログ・ファイルを指定します。
- ***num_files*** オプションの INTEGER パラメータを使用して、分析する要求ログ・ファイルの数を指定します。

備考

このプロシージャは、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、コマンド・プロンプトで -zo 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 権限が必要です。

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」 905 ページ
- 「sa_statement_text システム・プロシージャ」 964 ページ
- 「sa_server_option システム・プロシージャ」 948 ページ

sa_get_request_times システム・プロシージャ

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*** オプションの LONG VARCHAR パラメータを使用して、要求ロギングのファイル名を指定します。
- ***conn_id*** オプションの UNSIGNED INTEGER パラメータを使用して、接続の ID 番号を指定します。
- ***first_file*** オプションの INTEGER パラメータを使用して、分析する最初のファイルを指定します。
- ***num_files*** オプションの INTEGER パラメータを使用して、分析する要求ログ・ファイルの数を指定します。

備考

このプロシージャは、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 に渡します。

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他のアクションが実行される場合、時間値が大きいように見えますが、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、コマンド・プロンプトで -zo 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 権限が必要です。

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* 内の要求の実行回数を取得します。

```
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」 904 ページ
- 「sa_statement_text システム・プロシージャ」 964 ページ
- 「sa_server_option システム・プロシージャ」 948 ページ

sa_get_server_messages システム・プロシージャ

サーバのメッセージ・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。

構文

sa_get_server_messages(*first_line* **)**

引数

- ***first_line*** INTEGER パラメータを使用して、サーバ・メッセージの表示を開始する行番号を指定します。

結果セット

カラム名	データ型	説明
line_num	INTEGER	サーバ・メッセージの行番号。
message_text	VARCHAR(255)	サーバ・メッセージのテキスト。
message_time	TIMESTAMP	メッセージの時刻。

備考

このプロシージャは、表示する最初の行番号を指定する INTEGER パラメータを受け取り、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

次の例では、sa_get_server_messages システム・プロシージャを使用して、サーバ・メッセージ・ウィンドウの 16 行目から開始する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 2000 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。	...
...	...	...

sa_http_header_info システム・プロシージャ

HTTP ヘッダ名と値を返します。

構文

sa_http_header_info([*header_parm*] **)**

引数

- ***header_parm*** オプションの VARCHAR(255) パラメータを使用して、HTTP ヘッダ名を指定します。

結果セット

カラム名	データ型	説明
Name	VARCHAR(255)	HTTP ヘッダ名。
Value	LONG VARCHAR	HTTP ヘッダの値。

備考

sa_http_header_info システム・プロシージャは、HTTP ヘッダ名と値を返します。オプションのパラメータを使用してヘッダ名を指定しない場合、結果セットにはすべてのヘッダの値が格納されます。

このプロシージャは、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

- 「SQL Anywhere Web サービス」 『SQL Anywhere サーバ - プログラミング』
- 「sa_http_variable_info システム・プロシージャ」 908 ページ

sa_http_variable_info システム・プロシージャ

HTTP 変数名と値を返します。

構文

sa_http_variable_info([*variable_parm*] **)**

引数

- ***variable_parm*** オプションの VARCHAR(255) パラメータを使用して、HTTP 変数名を指定します。

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・プロシージャは、HTTP 変数名と値を返します。オプションのパラメータ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。

このプロシージャは、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

- 「SQL Anywhere Web サービス」『SQL Anywhere サーバ - プログラミング』
- 「sa_http_header_info システム・プロシージャ」 908 ページ

sa_index_density システム・プロシージャ

データベース・インデックス内の断片化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*** オプションの CHAR(128) パラメータを使用して、テーブル名を指定します。
- ***owner_name*** オプションの CHAR(128) パラメータを使用して、所有者名を指定します。

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。

カラム名	データ型	説明
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	NUMERIC(8.6)	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 ー 1 の間の小数。

備考

データベース管理者は、このプロシージャを使ってデータベースのインデックス内の断片化の程度に関する情報を取得できます。

このプロシージャは、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、リーフ・ページの数、インデックスの密度で構成される結果セットを返します。

パラメータを指定しない場合、すべてのテーブルの情報が表示されます。それ以外の場合は、指定したテーブルだけが検査されます。

多数のリーフ・ページを持つインデックスの場合は、density 値を大きくすることをおすすめします。

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

- 「インデックスの断片化削減」 『SQL Anywhere サーバ - SQL の使用法』

例

次の例では、sa_index_density システム・プロシージャを使用して、データベース・インデックス内の断片化の量をまとめた結果セットを返します。

```
CALL sa_index_density( );
```

TableName	TableId	IndexName	...	Density
Products	436	Products	...	0.012451
...	...	...	...	...

sa_index_levels システム・プロシージャ

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*** オプションの CHAR(128) パラメータを使用して、テーブル名を指定します。
- ***owner_name*** オプションの CHAR(128) パラメータを使用して、所有者名を指定します。

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
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 権限が必要です。

関連する動作

なし。

参照

- 「CREATE INDEX 文」 414 ページ
- 「インデックスの使用」 『SQL Anywhere サーバ - SQL の使用法』

例

次の例では、sa_index_levels システム・プロシージャを使用して、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_java_loaded_classes システム・プロシージャ

データベース Java 仮想マシン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 Java 仮想マシン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 Java 仮想マシンによって現在ロードされている Java クラスのすべての名前で構成される結果セットを返します。

仮想マシンを初めて呼び出すと、いくつかのクラスがロードされます。データベース機能の Java 機能を使用したことがない状態で sa_java_loaded_classes を呼び出すと、この一連のクラスが返されます。

このプロシージャは、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 権限が必要です。

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 - プログラミング』.

sa_load_cost_model システム・プロシージャ

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*** CHAR(1024) パラメータを使用して、ロードするコスト・モデル・ファイルの名前を指定します。

備考

オプティマイザは、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・サーバは、各データベースのコスト・モデルを保守します。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、同様のハードウェアにあるデータベースの場合、生成する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・プロシージャを使用してファイルに保存されます。「sa_unload_cost_model システム・プロシージャ」 969 ページを参照してください。

パーミッション

DBA 権限が必要です。

関連する動作

データベース・サーバは新規コスト・モデルをロードした後に COMMIT を実行します。

参照

- 「ALTER DATABASE 文」 303 ページ
- 「sa_unload_cost_model システム・プロシージャ」 969 ページ

- 「クエリの最適化と実行」 『SQL Anywhere サーバ - 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 );
```

sa_locks システム・プロシージャ

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*** INTEGER パラメータを使用して、接続 ID を指定します。このプロシージャは、指定された接続に関するロック情報だけを返します。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*** この CHAR(128) パラメータを使用して、ユーザ ID を指定します。プロシージャは、指定したユーザが所有するテーブルに関する情報のみを返します。creator パラメータのデフォルト値は NULL です。このパラメータが 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* パラメータを指定すると、現在のユーザ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*** この CHAR(128) パラメータを使用して、テーブル名を指定します。プロシージャは、指定したテーブルのロック情報だけを返します。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*** この INTEGER パラメータを使用して、情報が返されるロックの最大数を指定します。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	接続の ID 番号。

カラム名	データ型	説明
user_id	CHAR(128)	接続 ID を通じて接続されたユーザ。
table_type	CHAR(6)	テーブルの種類 (BASE または GBLTMP)。
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。「ロックできるオブジェクト」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
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。「スキーマ・ロック」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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) Write (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、カーソル安定性 (Position) のために読み取りロックが短期か、COMMIT/ROLLBACK (Transaction) まで保持される長期かを指定します。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表示されるときに作成されます。「挿入時のロック」『SQL Anywhere サーバ - SQL の使用法』を参照してください。テーブルで作成されるすべての代理エントリについて一意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一意です。「ロー・ロック」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
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 - 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)	ほとんどの場合、位置のロックは、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 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、または連続スキャン) に保持されます。「位置ロック」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

位置のロックは、連続テーブル・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、位置のロックが連続スキャンに関係しているかどうかを示します。連続スキャンのために位置のロックを保持する場合、index_id カラムは NULL です。位置の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。位置のロックがすべてのインデックス全体でスキャンが保持される場合、インデックスの ID 値は -1 です。

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

- 「ロックの仕組み」『SQL Anywhere サーバ - SQL の使用法』
- 「SYSIDX システム・ビュー」 796 ページ

例

このシステム・プロシージャの例、および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」『SQL Anywhere サーバ - 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

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*** この CHAR(30) パラメータを使用して、作成されるオブジェクトのタイプを指定します。
- ***objname*** この CHAR(128) パラメータを使用して、作成されるオブジェクトの名前を指定します。
- ***owner*** オプションの CHAR(128) パラメータを使用して、作成されるオブジェクトの所有者を指定します。デフォルト値は CURRENT USER です。

- ***tabname*** この CHAR(128) パラメータは、objtype が 'trigger' である場合にのみ必要です。トリガ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やコマンド・ファイルで特に便利です。このようなスクリプトでは、最初に CREATE 文を実行しますが、その後は ALTER 文を実行することが共通の問題です。このプロシージャを使用すると、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・ビューにクエリを実行する必要がなくなります。

使用するには、このプロシージャ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」 311 ページ
- 「ALTER FUNCTION 文」 313 ページ
- 「ALTER PROCEDURE 文」 319 ページ
- 「ALTER TRIGGER 文」 345 ページ
- 「ALTER VIEW 文」 346 ページ
- 「ALTER SERVICE 文」 327 ページ

例

次の文は、スケルトン・プロシージャ定義が作成されていることを確認し、プロシージャを定義し、プロシージャ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コマンド・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_info システム・プロシージャ

指定した実体化ビュー (Materialized View) 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*** オプションの CHAR(128) パラメータを使用して、情報を返す実体化ビュー (Materialized View) の名前を指定します。
- ***owner_name*** オプションの CHAR(128) パラメータを使用して、実体化ビュー (Materialized View) の所有者を指定します。

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実体化ビュー (Materialized View) に関する情報が返されます。

owner_name が指定されない場合、指定したビュー名と一致する実体化ビュー (Materialized View) のみが記述されます。このプロシージャには、DBA 権限が必要です。または、プロシージャに所有されている DBO の許可を実行します。

sa_materialized_view_info システム・プロシージャは、次の情報を返します。

カラム名	データ型	説明
OwnerName	CHAR(128)	ビューの作成者。
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。 ◆ D - ユーザが無効化 ◆ N - 更新なし ◆ E - 最新のリフレッシュ試行時にエラー ◆ F - 基礎となるデータが最後の更新以来、変更されていない場合 (新しい) ◆ S - 基礎となるデータが最後の更新以来、変更された場合 (古い)
ViewLastRefreshed	TIMESTAMP	ビューを最後に更新した時間。ビューにデータがない場合、この値は NULL です (未初期化)。
DataLastModified	TIMESTAMP	古いビューの場合、基礎となるデータを修正した最後の時間。

カラム名	データ型	説明
AvailForOptimization	CHAR(1)	オプティマイザが使用するビューの使用可能性に関する情報。 ◆ Y - ビューはオプティマイザから使用できます ◆ D - 無効なオプティマイザが使用 ◆ N - 更新が実行されたか失敗したために、データが含まれません ◆ I - 何らかの内部的な理由からビューを使用できません ◆ O - 現在の接続について互換性がないオプション値

このプロシージャは、実体化ビュー (Materialized View) の定義に問題があるためにオプティマイザが考慮しない実体化ビュー (Materialized View) のリストを決定する場合に便利です。このような実体化ビュー (Materialized View) の場合、AvailForOptimization 値は I です。実体化ビュー (Materialized View) の定義に関する制限の詳細については、「実体化ビュー (Materialized View) を管理するときの制限」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

パーミッション

DBA 権限が必要です。

関連する動作

指定した実体化ビュー (Materialized View) のすべてのメタデータと依存性は、サーバのキャッシュにロードされます。

sa_migrate システム・プロシージャ

SQL Anywhere データベースにリモート・テーブル・セットを移行します。

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*** この VARCHAR(128) パラメータを使用して、SQL Anywhere ターゲット・データベースで、移行後のテーブルを所有するユーザを指定します。GRANT CONNECT

文を使用してこのユーザを作成します。このパラメータの値は必須です。「GRANT 文」 565 ページを参照してください。

- ***server_name*** この VARCHAR(128) パラメータを使用して、リモート・データベースへの接続に使用するリモート・サーバの名前を指定します。CREATE SERVER 文を使用してこのサーバを作成します。このパラメータの値は必須です。「CREATE SERVER 文」 444 ページを参照してください。
- ***table_name*** 単一のテーブルを移行する場合は、この VARCHAR(128) パラメータ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移行する場合は、この VARCHAR(128) パラメータ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に NULL を指定しないでください。
- ***database_name*** この VARCHAR(128) パラメータを使用して、リモート・データベースの名前を指定します。リモート・サーバ上の 1 つのデータベースのみからテーブルを移行する場合は、データベース名を指定します。それ以外の場合は、このパラメータに NULL (デフォルト) を入力します。
- ***migrate_data*** オプションの BIT パラメータを使用して、リモート・テーブルのデータを移行するかどうかを指定します。このパラメータは、0 (データを移行しない) または 1 (データを移行する) に設定できます。デフォルトでは、データが移行します。 (1)
- ***drop_proxy_tables*** オプションの BIT パラメータを使用して、移行の完了後に、移行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します。このパラメータは、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*** オプションの BIT パラメータを使用して、外部キー・マッピングが移行されるかどうかを指定します。このパラメータは、0 (外部キー・マッピングを移行しない) または 1 (外部キー・マッピングを移行する) に設定できます。デフォルトでは、外部キー・マッピングが移行します (1)。

備考

このプロシージャを使用して、Oracle、DB2、SQL Server、SQL Enterprise、Adaptive Server Enterprise、SQL Anywhere のデータベースから SQL Anywhere テーブルに移行できます。このプロシージャでは、1 回の操作だけで、指定したサーバから外部キー・マッピングを含めたリモート・テーブル・セットを移行できます。sa_migrate システム・プロシージャは、次のシステム・プロシージャを呼び出します。

- 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

移行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・プロシージャを使用できます。たとえば、別のユーザ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移行する場合は、sa_migrate を使用すると外部キー関係を保持できません。

テーブルを移行するためには、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。「CREATE SERVER 文」 444 ページと「CREATE EXTERNLOGIN 文」 406 ページを参照してください。

base_table_owner と *server_name* の各パラメータを指定するだけで、リモート・データベースのすべてのテーブルを SQL Anywhere データベースに移行できます。ただし、これら 2 つのパラメータ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、移行する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移行します。その際、sa_migrate プロシージャを呼び出すたびに *base_table_owner* パラメータと *owner_name* パラメータを指定します。

警告
table_name と *owner_name* の両方のパラメータに NULL を指定しないでください。*table_name* パラメータと *owner_name* パラメータの両方に 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移行します。また、リモート・データベースで、テーブルの所有者が異なっていても名前が同じ場合、それらのテーブルはターゲット・データベースに移行すると 1 人の所有者に属します。一度に移行するテーブルは、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 へのデータベースの移行」 『SQL Anywhere サーバ - SQL の使用法』
- 「sa_migrate_create_remote_table_list システム・プロシージャ」 926 ページ
- 「sa_migrate_create_tables システム・プロシージャ」 928 ページ
- 「sa_migrate_data システム・プロシージャ」 929 ページ
- 「sa_migrate_create_remote_fks_list システム・プロシージャ」 925 ページ
- 「sa_migrate_create_fks システム・プロシージャ」 924 ページ
- 「sa_migrate_drop_proxy_tables システム・プロシージャ」 930 ページ

例

次の文は、リモート・データベースからユーザ 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移行し、リモート・テーブル内のデータを移行します。また、移行の完了時

にプロキシ・テーブルを削除します。この例では、移行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 remote_a に属するテーブルのみをリモート・データベースから移行します。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の local_a に所属します。移行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

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*** この VARCHAR(128) パラメータを使用して、SQL Anywhere ターゲット・データベースで、移行後の外部キーを所有するユーザを指定します。異なるユーザに属するテーブルを移行する場合は、移行するテーブルを所有するユーザごとに、このプロシージャを実行します。*i_table_owner* は、GRANT CONNECT 文を使用して作成されます。このパラメータの値は必須です。「GRANT 文」 565 ページを参照してください。

備考

このプロシージャは、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、外部キーを移行する必要のあるテーブルを所有しているユーザごとに、このプロシージャを実行します。

注意
このシステム・プロシージャは、その他の移行システム・プロシージャ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。

参照

- 「SQL Anywhere へのデータベースの移行」『SQL Anywhere サーバ - SQL の使用法』
- 「sa_migrate システム・プロシージャ」 921 ページ
- 「sa_migrate_create_remote_table_list システム・プロシージャ」 926 ページ
- 「sa_migrate_create_tables システム・プロシージャ」 928 ページ
- 「sa_migrate_data システム・プロシージャ」 929 ページ
- 「sa_migrate_create_remote_fks_list システム・プロシージャ」 925 ページ
- 「sa_migrate_drop_proxy_tables システム・プロシージャ」 930 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。Adaptive Sever Anywhere ローカル・データベースでは、外部キーはユーザ 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

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*** この VARCHAR(128) パラメータを使用して、リモート・データベースへの接続に使用するリモート・サーバの名前を指定します。リモート・サーバは、CREATE SERVER 文で作成します。このパラメータの値は必須です。「CREATE SERVER 文」 444 ページを参照してください。

備考

このプロシージャは、リモート・データベースから移行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、移行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・プロシージャは、他のいくつかの移行システム・プロシージャと組み合わせて使用されます。sa_migrate_create_fks システム・プロシージャの備考部分には、移行プロジェクトのリストと、実行順序が記載されています。「sa_migrate_create_fks システム・プロシージャ」 924 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベースの移行」 『SQL Anywhere サーバ - SQL の使用法』
- 「sa_migrate システム・プロシージャ」 921 ページ
- 「sa_migrate_create_remote_table_list システム・プロシージャ」 926 ページ
- 「sa_migrate_create_tables システム・プロシージャ」 928 ページ
- 「sa_migrate_data システム・プロシージャ」 929 ページ
- 「sa_migrate_create_fks システム・プロシージャ」 924 ページ
- 「sa_migrate_drop_proxy_tables システム・プロシージャ」 930 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

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*** この VARCHAR(128) パラメータを使用して、リモート・データベースへの接続に使用するリモート・サーバの名前を指定します。リモート・サーバは、CREATE SERVER 文で作成します。このパラメータの値は必須です。「CREATE SERVER 文」 444 ページを参照してください。
- ***i_table_name*** オプションの VARCHAR(128) パラメータを使用して、移行するテーブルの名前、またはすべてのテーブルを移行する場合は 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に NULL を指定しないでください。
- ***i_owner_name*** オプションの VARCHAR(128) パラメータを使用して、リモート・データベース上の移行するテーブルを所有するユーザ、またはすべてのテーブルを移行する場合は 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に NULL を指定しないでください。
- ***i_database_name*** オプションの VARCHAR(128) パラメータを使用して、テーブルの移行元のリモート・データベースの名前を指定します。このパラメータのデフォルトは NULL です。Adaptive Server Enterprise と Microsoft SQL Server データベースからテーブルを移行する場合は、データベース名を指定します。

備考

このプロシージャは、リモート・データベースから移行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、移行しないリモート・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、移行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、移行するテーブルを所有するユーザごとにこのプロシージャを実行します。

または、sa_migrate システム・プロシージャを使用すると、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に NULL を指定しないでください。
i_table_name パラメータと *i_owner_name* パラメータの両方に 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移行します。また、リモート・データベースで、テーブルの所有者が異なっていても名前が同じ場合、それらのテーブルはターゲット・データベースに移行すると 1 人の所有者に属します。一度に移行するテーブルは、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・プロシージャは、他のいくつかの移行システム・プロシージャと組み合わせて使用されます。sa_migrate_create_fks システム・プロシージャの備考部分には、移行プロジェクトのリストと、実行順序が記載されています。「sa_migrate_create_fks システム・プロシージャ」 924 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベースの移行」『SQL Anywhere サーバ - SQL の使用法』
- 「sa_migrate システム・プロシージャ」 921 ページ
- 「sa_migrate_create_tables システム・プロシージャ」 928 ページ
- 「sa_migrate_data システム・プロシージャ」 929 ページ
- 「sa_migrate_create_remote_fks_list システム・プロシージャ」 925 ページ
- 「sa_migrate_create_fks システム・プロシージャ」 924 ページ
- 「sa_migrate_drop_proxy_tables システム・プロシージャ」 930 ページ

例

次の文は、リモート・データベース上のユーザ 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

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*** この VARCHAR(128) パラメータを使用して、SQL Anywhere ターゲット・データベースで、移行後のテーブルを所有するユーザを指定します。このユーザは、GRANT CONNECT 文を使用して作成します。このパラメータの値は必須です。「GRANT 文」 565 ページを参照してください。

備考

このプロシージャは、dbo.migrate_remote_table_list テーブル (sa_migrate_create_remote_table_list プロシージャ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が所有します。このプロシージャは、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、移行後のすべてのテーブルの所有者を同じにしない場合は、移行するテーブルを所有するユーザごとに、sa_migrate_create_remote_table_list プロシージャと sa_migrate_create_tables プロシージャを実行します。

または、sa_migrate システム・プロシージャを使用すると、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・プロシージャは、他のいくつかの移行システム・プロシージャと組み合わせて使用されます。sa_migrate_create_fks システム・プロシージャの備考部分には、移行プロジェクトのリストと、実行順序が記載されています。「sa_migrate_create_fks システム・プロシージャ」 924 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベースの移行」『SQL Anywhere サーバ - SQL の使用法』
- 「sa_migrate システム・プロシージャ」 921 ページ
- 「sa_migrate_create_remote_table_list システム・プロシージャ」 926 ページ
- 「sa_migrate_data システム・プロシージャ」 929 ページ
- 「sa_migrate_create_remote_fks_list システム・プロシージャ」 925 ページ
- 「sa_migrate_create_fks システム・プロシージャ」 924 ページ
- 「sa_migrate_drop_proxy_tables システム・プロシージャ」 930 ページ

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 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

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移行します。

構文

sa_migrate_data(*i_table_owner* **)**

引数

- ***i_table_owner*** この VARCHAR(128) パラメータを使用して、SQL Anywhere ターゲット・データベースで、移行後のテーブルを所有するユーザを指定します。このユーザは、GRANT CONNECT 文を使用して作成します。このパラメータの値は必須です。「GRANT 文」 565 ページを参照してください。

備考

このプロシージャは、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に属するテーブルのデータを移行します。

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、データを移行するテーブルを所有しているユーザごとに、このプロシージャを実行します。

または、sa_migrate システム・プロシージャを使用すると、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・プロシージャは、他のいくつかの移行システム・プロシージャと組み合わせて使用されます。sa_migrate_create_fks システム・プロシージャの備考部分には、移行プロジェクトのリストと、実行順序が記載されています。「sa_migrate_create_fks システム・プロシージャ」 924 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベースの移行」 『SQL Anywhere サーバ - SQL の使用法』
- 「sa_migrate システム・プロシージャ」 921 ページ
- 「sa_migrate_create_remote_table_list システム・プロシージャ」 926 ページ
- 「sa_migrate_create_tables システム・プロシージャ」 928 ページ
- 「sa_migrate_create_remote_fks_list システム・プロシージャ」 925 ページ
- 「sa_migrate_create_fks システム・プロシージャ」 924 ページ
- 「sa_migrate_drop_proxy_tables システム・プロシージャ」 930 ページ

例

次の文は、ユーザ local_a に属するテーブルのデータを SQL Anywhere ターゲット・データベースに移行します。

```
CALL sa_migrate_data( 'local_a' );
```

sa_migrate_drop_proxy_tables システム・プロシージャ

移行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*** この VARCHAR(128) パラメータを使用して、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を指定します。このユーザは、GRANT CONNECT 文を使用して作成します。このパラメータの値は必須です。「GRANT 文」 565 ページを参照してください。

備考

このプロシージャは、移行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、移行後のテーブルをすべて同じユーザが所有しない場合、ユーザごとにこのプロシージャを呼び出して、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・プロシージャは、他のいくつかの移行システム・プロシージャと組み合わせて使用されます。sa_migrate_create_fks システム・プロシージャの備考部分には、移行プロジェクトのリストと、実行順序が記載されています。「sa_migrate_create_fks システム・プロシージャ」 924 ページを参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere へのデータベースの移行」 『SQL Anywhere サーバ - SQL の使用法』
- 「sa_migrate システム・プロシージャ」 921 ページ
- 「sa_migrate_create_remote_table_list システム・プロシージャ」 926 ページ
- 「sa_migrate_create_tables システム・プロシージャ」 928 ページ
- 「sa_migrate_data システム・プロシージャ」 929 ページ
- 「sa_migrate_create_remote_fks_list システム・プロシージャ」 925 ページ
- 「sa_migrate_create_fks システム・プロシージャ」 924 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 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_performance_diagnostics システム・プロシージャ

データベース・サーバ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。
Name	VARCHAR(255)	接続の名前。
Userid	VARCHAR(255)	接続のユーザ ID。
DBNumber	INT	データベースの ID 番号。
LoginTime	TIMESTAMP	接続が確立された日付と時刻。
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間。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。
LastReqTime	TIMESTAMP	指定の接続に対する最後の要求が開始した時間。
ReqType	VARCHAR(255)	最終要求のタイプ。
ReqStatus	VARCHAR(255)	要求のステータス。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、ワーカ・スレッドの解放を待っている。 ◆ **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・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。
ReqTimeUnscheduled	DOUBLE	予定外に要した時間。
ReqTimeActive	DOUBLE	要求を処理するときの待ち時間。

カラム名	データ型	説明
ReqTimeBlockIO	DOUBLE	I/O 終了までの待ち時間。
ReqTimeBlockLock	DOUBLE	ロックの待ち時間。
ReqTimeBlockContention	DOUBLE	アトミック・アクセスの待ち時間。
ReqCountUnscheduled	INT	スケジューリングを待った回数。
ReqCountActive	INT	処理された要求の数。
ReqCountBlockIO	INT	I/O 終了を待った回数。
ReqCountBlockLock	INT	ロックを待った回数。
ReqCountBlockContention	INT	アトミック・アクセスを待った回数。
LastIdle	INT	要求間のチックの数。
BlockedOn	INT	現在の接続が制限されていない場合は 0。ブロックされている場合は、ロック矛盾によってブロックされる接続の数。
UncommitOp	INT	コミットされていないオペレーションの数。
CurrentProcedure	VARCHAR(255)	接続が現在実行しているプロシージャ。接続でネストされたプロシージャ・コールが実行されている場合は、現在のプロシージャの名前を返します。実行されているプロシージャがない場合は、空の文字列を返します。
EventName	VARCHAR(255)	接続でイベント・ハンドラが実行されている場合の関連するイベント名。それ以外の場合、結果は NULL。
CurrentLineNumber	INT	接続で実行されているプロシージャまたは複合文の現在の行番号。実行されているプロシージャの名前は 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。
LastStatement	LONG VARCHAR	現在の接続で最後に作成された SQL 文。
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。

カラム名	データ型	説明
AppInfo	LONG VARCHAR	接続を確立したクライアントに関する情報。HTTP 接続では、ブラウザの情報が含まれています。jConnect または 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、または ADO.NET のいずれかです。
LockCount	INT	接続で保持されているロックの数。
SnapshotCount	INT	接続に関連付けられているスナップショットの数。

備考

sa_performance_diagnostics システム・プロシージャは、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・サーバでオンにする必要があります。文の記録をオンにするには、データベース・サーバの起動時に -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 権限が必要です。

関連する動作

なし。

参照

- 「-zt オプション」 『SQL Anywhere サーバ - データベース管理』
- 「sa_performance_statistics システム・プロシージャ」 935 ページ
- 「sa_server_option システム・プロシージャ」 948 ページ

例

次のクエリを実行すると、データベース・サーバ要求が終了するまでの待ち時間が長期の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

データベース・サーバ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。
ConnNumber	INT	接続 ID を示す INTEGER。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。
PropName	VARCHAR (255)	接続プロパティの名前。
値	INT	接続プロパティの値。

備考

sa_performance_statistics システム・プロシージャは、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・サーバでオンにする必要があります。文の記録をオンにするには、データベース・サーバの起動時に -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 権限が必要です。

関連する動作

なし。

参照

- 「-zt オプション」 『SQL Anywhere サーバ - データベース管理』
- 「sa_performance_diagnostics システム・プロシージャ」 931 ページ
- 「sa_server_option システム・プロシージャ」 948 ページ

例

次の例は、すべてのパフォーマンス統計情報を *dump_stats.txt* というテキスト・ファイルにアンロードします。

```
UNLOAD
  SELECT CURRENT TIMESTAMP, *
```



```
FROM sa_performance_statistics()
TO 'dump_stats.txt'
APPEND ON
```

sa_procedure_profile システム・プロシージャ

データベースで実行されたプロシージャ、関数、イベント、またはトリガ内の各行について、実行時間に関する情報をレポートします。このプロシージャは、Sybase Central の [プロファイル] タブと同じ情報を提供します。

構文

sa_procedure_profile(
  [*filename*
  [, *save_to_file*]]
)

引数

- ***filename*** オプションの LONG VARCHAR(128) パラメータを使用して、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*** オプションの INT(1) パラメータを使用して、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します。

結果セット

カラム名	データ型	説明
object_type	CHAR(1)	オブジェクトの型。使用できるオブジェクト型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、関数、イベント、またはトリガの名前。object_type が **C** または **D** の場合、これはシステム・トリガ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)	トリガに対応するテーブル (他のオブジェクト型の場合、値は NULL)。
line_num	UNSIGNED INTEGER	プロシージャ内の行番号。
executions	UNSIGNED INTEGER	行の実行回数。
millisecs	UNSIGNED INTEGER	行の実行時間 (ミリ秒単位)。
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・トリガの外部テーブルを所有するデータベース・ユーザ。

カラム名	データ型	説明
foreign_table	CHAR(128)	システム・トリガの外部テーブルの名前。

備考

このプロシージャは、次の用途で使用できます。

- **詳細なプロシージャ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、プロシージャを呼び出すだけです。
- **詳細なプロシージャ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、オフにすることもできます)。この方法でプロシージャを使用すると、ロードしたファイルは、プロシージャを実行しているデータベースと同じデータベースが作成します。それ以外の場合、結果は使用できません。

結果セットには、プロシージャ、トリガ、関数、イベント内の個々の行に関する実行回数の情報だけでなく、行が使用するプロシージャ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を微調整します。

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。「プロシージャ・プロファイリングの有効化」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

結果セットの object_type フィールドは、次のようになります。

- **P**　ストアド・プロシージャ
- **F**　関数
- **E**　イベント
- **T**　トリガ
- **C**　ON UPDATE システム・トリガ
- **D**　ON DELETE システム・トリガ

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 プロシージャを使用します。

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

- 「sa_server_option システム・プロシージャ」 948 ページ
- 「sa_procedure_profile_summary システム・プロシージャ」 938 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、関数、イベント、またはトリガ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のプロファイル情報を返します。また、*detailedinfo.txt* というファイルに保存します。

```
CALL sa_procedure_profile( "detailedinfo.txt", 1 );
```

次の文はいずれも、*detailedinfoOLD.txt* というファイルから詳細なプロシージャ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

データベースで実行されたすべてのプロシージャ、関数、イベント、またはトリガ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*** オプションの LONG VARCHAR(128) パラメータを使用して、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*** オプションの INT(1) パラメータを使用して、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します。

結果セット

カラム名	データ型	説明
object_type	CHAR(1)	オブジェクトの型。使用できるオブジェクト型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。

カラム名	データ型	説明
object_name	CHAR(128)	ストアド・プロシージャ、関数、イベント、またはトリガの名前。
owner_name	CHAR(128)	オブジェクトの所有者。
table_name	CHAR(128)	トリガに対応するテーブル (他のオブジェクト型の場合、値は NULL)。
executions	UNSIGNED INTEGER	各プロシージャの実行回数。
millisecs	UNSIGNED INTEGER	プロシージャの実行時間 (ミリ秒単位)。
foreign_owner	CHAR(128)	システム・トリガの外部テーブルを所有するデータベース・ユーザ。
foreign_table	CHAR(128)	システム・トリガの外部テーブルの名前。

備考

このプロシージャは、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、オフにすることもできます)。この方法でプロシージャを使用すると、ロードしたファイルは、プロシージャを実行しているデータベースと同じデータベースが作成します。それ以外の場合、結果は使用できません。

このプロシージャは、ストアド・プロシージャ、関数、イベント、トリガ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、遅いプロシージャ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。「プロシージャ・プロファイリングの有効化」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

結果セットの object_type フィールドは、次のようになります。

- **P** ストアド・プロシージャ
- **F** 関数
- **E** イベント
- **T** トリガ
- **S** システム・トリガ

- **C** ON UPDATE システム・トリガ
- **D** ON DELETE システム・トリガ

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 プロシージャを使用します。

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

- 「sa_server_option システム・プロシージャ」 948 ページ
- 「sa_procedure_profile システム・プロシージャ」 936 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、関数、イベント、またはトリガ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ます。また、*summaryinfo.txt* というファイルに保存します。

```
CALL sa_procedure_profile_summary( "summaryinfo.txt", 1 );
```

次の文はいずれも、*summaryinfoOLD.txt* というファイル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

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*** オプションの INTEGER パラメータを使用して、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します。0 と指定すると、カラム定義が失敗した場合、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。プロシージャは、コンパイルを必要とするものがなくまるまで、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。このプロシージャにゼロ以外のパラメータ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・プロシージャは、実体化ビュー (Materialized View) 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 権限が必要です。

関連する動作

非実体化ビュー (Non-materialized View) に VALID ステータスがない場合は、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

- 「ビューのステータス」 『SQL Anywhere サーバ - SQL の使用法』
- 「force_view_creation オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「ALTER VIEW 文」 346 ページ

sa_refresh_materialized_views システム・プロシージャ

初期化されていないステータスであるすべての実体化ビュー (Materialized View) を初期化します。

構文

sa_refresh_materialized_views([*ignore_errors*] **)**

引数

- ***ignore_errors*** オプションの INTEGER パラメータを使用して、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します。0 と指定すると、カラム定義が失敗した場合、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、または初期化または更新の最後の試行がエラーのために失敗したために、実体化ビュー (Materialized View) 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・プロシージャは、このようなすべての実体化ビュー (Materialized View) についてデータベースをスキャンし、初期化を試みます。プロシージャで実体化ビュー (Materialized View) 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、実体化ビュー (Materialized View) を初期化できます。

パーミッション

DBA 権限が必要です。

関連する動作

なし

参照

- 「REFRESH MATERIALIZED VIEW 文」 640 ページ
- 「実体化ビュー (Materialized View) のリフレッシュ」 『SQL Anywhere サーバ - SQL の使用法』

sa_remove_tracing_data システム・プロシージャ

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*** この INTEGER パラメータを使用して、データを削除するロギング・セッションの ID を指定します。

備考

指定した *log_session_id* にレコードがない場合、プロシージャに影響はありません。このプロシージャには戻り値がありません。

パーミッション

DBA 権限が必要です。

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、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」 761 ページ

sa_report_deadlocks システム・プロシージャ

データベース・サーバによって作成された内部バッファから、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 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。
wait_on	bigint	待機しているロックの名前。
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。

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・サーバは、デッドロックに関する情報を内部バッファに記録します。sa_report_deadlocks システム・プロシージャを使用して、ログ内の情報を表示できます。

パーミッション

DBA 権限が必要です。

関連する動作

なし。

参照

- 「log_deadlocks オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- 「ブロックされているユーザの判別」 『SQL Anywhere サーバ - SQL の使用法』

sa_reset_identity システム・プロシージャ

次の identity の値をテーブルに設定できます。このシステム・プロシージャを使用して、次に挿入されるローのオートインクリメント値を変更します。

構文

sa_reset_identity(
tbl_name,
owner_name,
new_identity
)

引数

- ***tbl_name*** この CHAR(128) パラメータを使用して、識別値をリセットするテーブルを指定します。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一意に識別する必要があります。
- ***owner_name*** この CHAR(128) パラメータを使用して、識別値をリセットするテーブルの所有者を指定します。
- ***new_identity*** この BIGINT パラメータを使用して、自動増分の開始値を指定します。

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。

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 が既に存在する場合、ローの挿入は失敗します。

*所有者*が指定されていないか NULL の場合、*tbl_name* はデータベースのテーブルを一意に識別する必要があります。

パーミッション

DBA 権限が必要です。

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。

例

次の文は次の ID 値を 101 にリセットします。

```
CALL sa_reset_identity( 'Employees', 'DBA', 100 );
```

sa_rowgenerator システム・プロシージャ

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*** オプションの INTEGER パラメータを使用して、開始値を指定します。デフォルト値は 0 です。
- ***rend*** オプションの INTEGER パラメータを使用して、終了値を指定します。デフォルト値は 100 です。
- ***rstep*** オプションの INTEGER パラメータを使用して、シーケンス値の増分を指定します。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、このプロシージャを使用できます。次のようなタスクには、sa_rowgenerator を使用できます。

- 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。

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)」 778 ページ

例

次のクエリは、現在の日付ごとに 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。該当する従業員がいない範囲もあり、その場合は警告「**集合関数では、Null 値は無視されます (-109)**」が生成されます。sa_rowgenerator プロシージャを使用すると、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、SQL Anywhere で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・プロシージャを使用して、0001-01-01 ー 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 ー 9999-12-31 (それぞれ SQL Anywhere でサポートされている最初の日と最後の日) 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 );
```

sa_save_trace_data システム・プロシージャ

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 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・プロシージャは、トレーシングが進行中でもエラーを返します。このシステム・プロシージャを使用するには、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・プロシージャを使用できます。また、プロシージャはトレーシング・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 権限が必要です。

関連する動作

オートコミット。

参照

- 「トレーシング・セッションの作成」『SQL Anywhere サーバ - SQL の使用法』
- 「診断トレーシング・テーブル」 761 ページ

sa_send_udp システム・プロシージャ

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*** この CHAR(254) パラメータを使用して、ホスト名または IP 番号を指定します。
- ***destPort*** この UNSIGNED SMALLINT パラメータを使用して、使用するポート番号を指定します。
- ***msg*** この LONG BINARY パラメータを使用して、指定されたアドレスに送信するメッセージを指定します。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

msg パラメータにバイナリ・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、変数を使用する必要があります。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;
```

このプロシージャを Mobile Link サーバで開始される同期とともに使用して、リスナ・ユーティリティ (*dblsn.exe*) を起動できます。リスナに通知する方法として sa_send_udp システム・プロシージャを使用する場合は、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リスナ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 権限が必要です。

関連する動作

なし。

参照

- 「sa_send_udp による Listener への通知」 『Mobile Link - サーバ起動同期』

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_option システム・プロシージャ

サーバ実行中に、サーバ・オプションを上書きします。

構文

sa_server_option(
opt,
val
)

引数

- ***opt*** この CHAR(128) パラメータを使用して、サーバ・オプション名を指定します。
- ***val*** この CHAR(128) パラメータを使用して、サーバ・オプションの新しい値を指定します。

備考

データベース管理者はこのプロシージャを使用して、データベース・サーバを再起動せずにデータベース・サーバ・オプションの一部を一時的に上書きできます。

このプロシージャを使用して変更されるオプション値は、サーバが停止するとデフォルト値にリセットされます。サーバ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・サーバの起動時に対応するサーバ・オプションがあればそれを指定できます。

次のオプション設定を変更できます。

オプション名	値の範囲	デフォルト	サーバ・オプション
CacheSizingStatistics	YES、NO	NO	「-cs サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
CollectStatistics	YES、NO	YES	「-k サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
ConnsDisabled	YES、NO	NO	
ConnsDisabledForDB	YES、NO	NO	
ConsoleLogFile	*filename*		「-o サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
ConsoleLogMaxSize	バイト単位での*ファイル・サイズ*		「-on サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
DatabaseCleaner	ON、OFF	ON	
DebuggingInformation	YES、NO	NO	「-z サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
IdleTimeout	整数 (分)	240	「-ti サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
LivenessTimeout	整数 (秒)	120	「-tl サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
ProcedureProfiling	YES、NO、RESET、CLEAR	NO	

オプション名	値の範囲	デフォルト	サーバ・オプション
ProfileFilterConn	*connection-id*		
ProfileFilterUser	*user-id*		
QuittingTime	有効な日付と時刻		「-tq サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
RememberLastPlan	YES、NO	NO	「-zp サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
RememberLastStatement	YES、NO	NO	「-zl サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
RequestFilterConn	*connection-id*, -1		
RequestFilterDB	*database-id*, -1		
RequestLogFile	*filename*		「-zo サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
RequestLogging	SQL、HOSTVARS、PLAN、PROCEDURES、TRIGGERS、OTHER、BLOCKS、REPLACE、ALL、YES、NONE、NO	NONE	「-zr サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
RequestLogMaxSize	バイト単位での*ファイル・サイズ*		「-zs サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
RequestLogNumFiles	整数		「-zn サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
RequestTiming	YES、NO	NO	「-zt オプション」『SQL Anywhere サーバ - データベース管理』
SecureFeatures	*feature-list*		「-sf サーバ・オプション」『SQL Anywhere サーバ - データベース管理』

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、サーバ・メッセージ・ウィンドウにキャッシュ情報を表示します。「-cs サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

CollectStatistics
YES に設定すると、データベース・サーバはパフォーマンス・モニタの統計情報を収集します。「-k サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

ConnsDisabled
YES に設定すると、データベース・サーバ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
サーバ・メッセージのウィンドウ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。「-o サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

ConsoleLogMaxSize
サーバ・メッセージのウィンドウ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)。出力ログ・ファイルが、sa_server_option システム・プロシージャまたは -on サーバ・オプションで指定されたサイズに達すると、ファイル名が変更されて拡張子 *.old* が追加されます (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)。出力ログ・ファイルが再起動します。「-on サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

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。「sa_clean_database システム・プロシージャ」 873 ページを参照してください。

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、サーバ・メッセージ・ウィンドウに表示されます。「-z サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

IdleTimeout
minutes で指定された時間の間、要求を送信しなかった TCP/IP または SPX の接続を切断します。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。「-ti サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

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／サーバの TCP/IP または SPX 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。「-tl サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

ProcedureProfiling
ストアド・プロシージャ、関数、イベント、トリガについてのプロシージャ・プロファイリングを制御します。プロシージャ・プロファイリングは、ストアド・プロシージャ、関数、イベント、トリガの実行所要時間を示します。Sybase Central の Database プロパティ・シートでもプロシージャ・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・プロファイリングを有効にします。
- **NO**　プロシージャ・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・カウンタをゼロに戻します。
- **CLEAR**　プロファイリング・カウンタをゼロに戻し、プロシージャ・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 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。「システム・プロシージャを使用したプロシージャ・プロファイリング」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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・サーバに指示します。接続フィルタが有効な場合、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が無効な場合、戻り値は -1 です。

ProfileFilterUser
データベース・サーバに、特定のユーザ 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・サーバに、指定された時間にサーバを停止するよう指示します。「-tq サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

RememberLastPlan property
接続で最後に実行された長いテキスト・プランをキャプチャするように、クエリのデータベース・サーバに指示します。この設定は、-zp サーバ・オプションによって制御されます。「-zp サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

LastPlanText 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 LastPlan の現在の値を取得できます。

```
SELECT CONNECTION_PROPERTY( 'LastPlanText' )
```

RememberLastStatement
データベース・サーバに、サーバ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・コールの場合、プロシージャ内の文ではなく、最も外側のプロシージャ・コールのみが表示されます。

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' )
```

詳細については、「サーバ・レベルのプロパティ」『SQL Anywhere サーバ - データベース管理』と「-zl サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

RememberLastStatement がオンの場合、次の文は指定された接続に対して最後に準備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 サーバ設定がオンになっている場合、ユーザはだれでも sa_conn_activity システム・プロシージャを呼び出すか、LastStatement 接続プロパティの値を取得することにより、他のユーザが最後に準備した 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 )
```

フィルタは、明示的にリセットされるか、データベース・サーバが停止するまで有効なままになります。フィルタ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ロギング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 )
```

フィルタは、明示的にリセットされるか、データベース・サーバが停止するまで有効なままになります。フィルタ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、サーバは要求をサーバ・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。「-zo サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。

RequestLogging
これは、データベース・サーバ・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されます。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。

- **SQL** 以下の項目のロギングを有効にします。
  - START DATABASE 文
  - STOP DATABASE 文
  - STOP ENGINE 文
  - 文の準備と実行
  - EXECUTE IMMEDIATE 文
  - オプション設定
  - COMMIT 文
  - ROLLBACK 文
  - PREPARE TO COMMIT 操作
  - 接続と接続解除
  - トランザクションの開始
  - DROP STATEMENT 文
  - カーソルの説明
  - カーソルを開く、閉じる、再開する
  - エラー
- **PLAN** クエリ・プランのロギングを有効にします (短期)。プロシージャのクエリ・プランは、プロシージャ (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。OTHRE を含めると、ログ・ファイルが急速に拡大し、サーバ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。

RequestLogging 設定の現在の値は、SELECT property('RequestLogging') を使用して検索できます。

詳細については、「-zr サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』と「サーバ・レベルのプロパティ」 『SQL Anywhere サーバ - 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-zs 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、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・プロシージャまたは -zs サーバ・オプションで指定されたサイズに達すると、ファイル名が変更されて拡張子 *.old* が追加されます (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)。要求ログ・ファイルが再起動します。「-zs サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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 オプションを使用して、保持する要求ログ・ファイルのコピーの数を指定できます。「-zn サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』を参照してください。

RequestTiming
データベースに、各接続のタイミング情報を保守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・サーバは各接続の累積タイマを保守します。このタイマは、接続が確立した状態でのサーバ・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

「接続レベルのプロパティ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

RequestTiming サーバ・プロパティがオンであると、追加のカウンタを保守するため、要求ごとに多少のオーバーヘッドがかかります。「-zt オプション」 『SQL Anywhere サーバ - データベース管理』と「sa_performance_diagnostics システム・プロシージャ」 931 ページを参照してください。

SecureFeatures
このデータベース・サーバで実行されるデータベースで無効にする機能を指定します。*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。有効な *feature-list* 値のリストについては、「-sf サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』を参照してください。

機能を無効または有効にするために行った変更は、直ちに有効になります。この設定は、sa_server_option システム・プロシージャを実行する接続に影響を与えません。

sa_server_option システム・プロシージャを使用して、現在のデータベース・サーバで実行されているすべてのデータベースの機能を有効または無効にするには、データベース・サーバの起動時に -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。secure_feature_key データベース・オプションを -sk 値に設定することで、現在の接続の全機能が有効になります。「-sk サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』と「secure_feature_key [データベース]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

パーミッション

DBA 権限が必要です。

関連する動作

なし。

例

次の文は、データベース・サーバ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 dbo.sa_server_option( 'RequestLogging', 'SQL+PROCEDURES+BLOCKS+PLAN+REPLACE' );
```

sa_set_http_header システム・プロシージャ

Web サービスによる結果内の HTTP ヘッダの設定を許可します。

構文

sa_set_http_header(
fldname,

val
)

引数

- ***fldname*** この CHAR(128) パラメータを使用して、HTTP ヘッダ・フィールドのいずれかの名前を含む文字列を指定します。
- ***val*** この LONG VARCHAR パラメータを使用して、指定されたパラメータに設定する値を指定します。

備考

特別なヘッダ・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、次のコマンドはステータス・コードを 404 Not Found に設定します。

```
CALL dbo.sa_set_http_header( '@HTTPSTATUS', '404' );
```

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。

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」 962 ページ

例

次の例は、Content-Type ヘッダ・フィールドを text/html に設定します。

```
CALL dbo.sa_set_http_header( 'Content-Type', 'text/html' );
```

sa_set_http_option システム・プロシージャ

Web サービスによる結果内の 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*** この CHAR(128) パラメータを使用して、HTTP オプションのいずれかの名前を含む文字列を指定します。

- ***val*** この LONG VARCHAR パラメータを使用して、指定されたオプションに設定する値を指定します。

備考

Web サービスを処理する文またはプロシージャ内でこのプロシージャを使用して、HTTP 結果セット内のオプションを設定します。

次のオプションがサポートされています。

- **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からクライアントの文字セット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。「自動文字セット変換の使用」『SQL Anywhere サーバ - プログラミング』を参照してください。
- **AcceptCharset** このオプションを使用して、XML、SOAP、HTTP Web サービスで使用されている文字セットについて、HTTP サーバのユーザ設定を指定します。このオプションの構文は、RFC2616 Hypertext Transfer Protocol の HTTP Accept-Charset 要求ヘッダ・フィールドの仕様に使用する構文に準拠します。つまり、許可されている値は、文字セットのラベルです。たとえば、sa_set_http_option('AcceptCharset', 'UTF-8, Shift_JIS') です。また、優先度を指定した品質値 (q) を文字セット・ラベルに含めることもできます。たとえば、sa_set_http_option('AcceptCharset', 'UTF-8; q=0.9, Shift_JIS') です。

  使用できる文字セットの最終リストは、クライアントの Accept-Charset HTTP 要求ヘッダ・フィールド値と、サーバの AcceptCharset オプションの値との共通部分です。クライアントが指定するほとんどの優先文字セットは可能であれば使用されるように、クライアント要求で指定されている品質値を順守します。次に、使用できる文字セットのリストを決定するときに可能なシナリオを示します。

  - クライアント要求が Accept-Charset 値を指定しない場合、サーバの AcceptCharset オプションが文字セットの決定に使用されます。
  - サーバの AcceptCharset オプションが指定されない場合、クライアントの Accept-Charset 要求ヘッダ・フィールド値は、文字セットの決定に使用されます。
  - サーバの AcceptCharset オプションもクライアントの Accept-Charset 要求ヘッダ・フィールド値も指定されない場合、データベースの文字セットが使用されます。SOAP 要求の場合、リクエストで使用されるコード化は、応答にも使用されます。
  - サーバの AcceptCharset オプションとクライアントの Accept-Charset 要求ヘッダ・フィールドの両方が指定されている場合、次のようになります。

    1. 一致が検出された場合は、その文字セットが使用されます。複数の一致が検出された場合、クライアントの Accept-Charset 要求ヘッダ・フィールドの最も高い q 値が使用されます。
    2. 一致が検出されない場合、クライアントの Accept-Charset 要求ヘッダ・フィールドに指定された文字セットが使用されます。

  SQL Anywhere では、可能であれば、プラス記号 (+) を指定してデータベースの文字セットを使用する場合のサーバの優先順位を指定できます。たとえば、sa_set_http_option

('AcceptCharset', 'UTF-8, +') です。指定した場合、クライアントの Accept-Charset 要求ヘッダ・フィールドに指定された文字セットのリストには、データベースの文字セットが含まれます。この文字セットは、品質値や検出された一致順序に関係なく、自動的に使用されます。これによって、文字セットの対話の要件を最小限にします。

- **sessionid** このオプションを使用して、HTTP セッションの名前を指定します。たとえば、sa_set_http_option('sessionid', 'my_app_session_1') のようにすると、HTTP セッションの ID が my_app_session_1 に設定されます。セッション ID には、NULL 以外の文字列を指定する必要があります。HTTP セッションの詳細については、「HTTP セッションの使用」 『SQL Anywhere サーバ - プログラミング』を参照してください。

パーミッション

なし。

関連する動作

なし。

参照

- 「SQL Anywhere Web サービス」 『SQL Anywhere サーバ - プログラミング』
- 「HTTP セッションの使用」 『SQL Anywhere サーバ - プログラミング』
- 「sa_set_http_header システム・プロシージャ」 956 ページ

sa_set_soap_header システム・プロシージャ

SOAP 応答の SOAP ヘッダの設定を許可します。このプロシージャは、SOAP Web サービスから呼び出されたストアド・プロシージャ内で使用されます。

構文

sa_set_soap_header(
fldname,
val
)

引数

- ***fldname*** この VARCHAR パラメータを使用して、特定のヘッダ・エントリを参照するときに使用する一意な文字列を指定します (*val* の localname と同じである必要はありません)。
- ***val*** この VARCHAR パラメータを使用して、トップ・レベルのヘッダ・エントリと、SOAP ヘッダ要素の範囲内にある子供の未加工 XML を指定します。

備考

このプロシージャ内で設定されたすべての SOAP ヘッダ・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・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

- 「SOAP_HEADER 関数 [SOAP]」 250 ページ
- 「NEXT_SOAP_HEADER 関数 [SOAP]」 210 ページ
- 「SOAP ヘッダの使用」 『SQL Anywhere サーバ - プログラミング』

例

次の例は、Hello に対する SOAP ヘッダのウェルカム メッセージを設定します。

```
sa_set_soap_header( 'welcome', '<welcome>Hello</welcome>' )
```

sa_set_tracing_level システム・プロシージャ

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*** この INTEGER パラメータを使用して、実行する診断トレーシングのレベルを指定します。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** トレーシングの基本レベルを設定します。
  - **2** トレーシングの中間レベルを設定します。
  - **3** トレーシングの高度レベルを設定します。
- ***specified_scope*** オプションの LONG VARCHAR パラメータを使用して、トレーシングのスコープを指定します。たとえば、USER、DATABASE、CONNECTION_NAME、TRIGGER などを指定します。
- ***specified_name*** オプションの LONG VARCHAR パラメータを使用して、*specified_scope* に示されたオブジェクトの識別子を指定します。

- ***do_commit*** オプションの TINYINT パラメータを使用して、このプロシージャで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します。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

このプロシージャは、sa_diagnostic_tracing_level テーブルにローを置換します。このとき、トレーシング・レベルと範囲は、プロシージャ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 権限が必要です。

関連する動作

なし。

参照

- 「トレーシング・レベルの選択」 『SQL Anywhere サーバ - SQL の使用法』
- 「診断トレーシングのスコープ」 『SQL Anywhere サーバ - SQL の使用法』
- 「sa_diagnostic_tracing_level テーブル」 775 ページ
- 「診断トレーシングを使用した詳細なアプリケーション・プロファイリング」 『SQL Anywhere サーバ - SQL の使用法』

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・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 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

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

複数の文のスナップショットが 1 つの接続に存在できます。文のスナップショット・レベルで実行されるネストされた文、またはインタ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を開始します。

通常、1 つの接続につき 1 つのトランザクションのスナップショットしかありません (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がある場合、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 権限が必要です。

関連する動作

なし。

参照

- 「sa_transactions システム・プロシージャ」 967 ページ
- 「スナップショット・アイソレーション」 『SQL Anywhere サーバ - SQL の使用法』

sa_split_list システム・プロシージャ

デリミタ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*** この LONG VARCHAR パラメータを使用して、分割する値を含む文字列を指定します。文字列は *delim* で区切られます。
- ***delim*** オプションの CHAR(10) パラメータを使用して、*str* の値を分割するときに使用するデリミタを指定します。デリミタは、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*** オプションの INTEGER パラメータを使用して、戻り値の最大長を指定します。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 文字の長さに切り詰められます。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	CHAR	文字列の値。必要に応じて、*maxlen* に切り詰められます。

備考

sa_split_list プロシージャは、sa_split_list の結果セットに対するクエリを制限する他のプロシージャ内で使用できます。

パーミッション

なし。

関連する動作

なし。

例

次の呼び出しは、個々の項目が結果セットの別の行に表示されるように、文字列「yellow##blue##red」を設定しています。

```
CALL sa_split_list( 'yellow##blue##red', '##', 3 );
```

line_num	row_value
1	yel
2	blu
3	red

次の例は、ProductsWithColor というプロシージャが作成されます。呼び出されると、ProductsWithColor プロシージャは sa_split_list を使用して、ユーザ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、ユーザが規定した色のいずれかに一致する各プロダクトの名前、説明、サイズ、色を返します。

次のプロシージャを呼び出した結果は、名前、説明、サイズ、青または白の全製品の色です。

```
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

CALL ProductsWithColor( 'white, blue' )
```

sa_statement_text システム・プロシージャ

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*** この LONG VARCHAR パラメータを使用して、SELECT 文を指定します。

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。

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」 905 ページ
- 「sa_get_request_profile システム・プロシージャ」 904 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

データベース・テーブルの断片化状況をレポートします。

構文

sa_table_fragmentation(
 [*tbl_name*
 [, *owner_name*]]
)

引数

- ***tbl_name*** オプションの CHAR(128) パラメータを使用して、断片化を調べるテーブルの名前を指定します。
- ***owner_name*** オプションの CHAR(128) パラメータを使用して、*tbl_name* の所有者を指定します。

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。
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、このプロシージャを使ってデータベースのテーブルの断片化状況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ら、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。「テーブルの断片化削減」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

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

- 「テーブルの断片化削減」 『SQL Anywhere サーバ - SQL の使用法』
- 「データベースの再構築」 『SQL Anywhere サーバ - SQL の使用法』
- 「REORGANIZE TABLE 文」 647 ページ

sa_table_page_usage システム・プロシージャ

データベース・テーブルのページの使用状況をレポートします。

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。
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。

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

- 「情報ユーティリティ (dbinfo)」 『SQL Anywhere サーバ - データベース管理』

sa_table_stats システム・プロシージャ

各テーブルから読み取ら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名。
table_name	CHAR(128)	テーブル名。
count	UNSIGNED BIGINT	SYSTEB 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の読み取り数。
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 プロシージャが返す各ローは、オプティマイザがページの統計情報を保守しているテーブルを記述しています。sa_table_stats プロシージャは、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取り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 プロシージャを使用して、ほとんどのディスク読み取りを生成しているテーブルを検索できます。プロシージャの結果は推定値なので、診断用途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、オプティマイザがテーブルのカウントを保守し始めてからディスクから読み取られたテーブルのページ数を指定します。これらの統計情報は、永続的にはデータベースに格納されません。これは、初めてメモリにロードされた後の、テーブルに関するアクティビティを示します。

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

- 「SYSTAB システム・ビュー」 823 ページ

sa_transactions システム・プロシージャ

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_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・サーバが追跡している間、トランザクションを一意に識別する 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。
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	コミットまたはロールバックされた場合に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 権限が必要です。

関連する動作

なし。

参照

- 「sa_snapshots システム・プロシージャ」 961 ページ
- 「スナップショット・アイソレーション」 『SQL Anywhere サーバ - SQL の使用法』

sa_unload_cost_model システム・プロシージャ

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*)

引数

- ***file_name*** CHAR(256) パラメータを使用して、データのアンロード先であるファイルの名前を指定します。システム・プロシージャを実行するデータベース・サーバなので、*file_name* はデータベース・サーバ・コンピュータ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・サーバの開始ディレクトリに相対するファイルを指定します。

備考

オプティマイザは、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・サーバは、各データベースのコスト・モデルを保守します。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・プロシージャは、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レプリケート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 権限が必要です。

ファイルを作成するには書き込み権限が必要です。

関連する動作

なし。

参照

- 「ALTER DATABASE 文」 303 ページ
- 「sa_load_cost_model システム・プロシージャ」 914 ページ
- 「クエリの最適化と実行」 『SQL Anywhere サーバ - 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_validate システム・プロシージャ

データベース内のすべてのテーブルを検証します。

構文

sa_validate(
 [*tbl_name*
 [, *owner_name*
 [, *check_type*]]]
)

引数

- ***tbl_name*** オプションの VARCHAR(128) パラメータを使用して、検証するテーブルの名前を指定します。このパラメータが NULL (デフォルト) の場合、sa_validate はすべてのテーブルを検証します。
- ***owner_name*** オプションの VARCHAR(128) パラメータを使用して、*tbl_name* の所有者を指定します。このパラメータが NULL (デフォルト) の場合、sa_validate はすべてのユーザのテーブルを検証します。
- ***check_type*** オプションの CHAR(10) パラメータを使用して、実行する検証のタイプを指定します。このパラメータが NULL (デフォルト) の場合、各テーブルは VALIDATE TABLE 文を使用して検証され、その他の検証は行われません。*check_type* の値は、次のいずれかです。
  - **express** WITH EXPRESS CHECK を使用してテーブルを検証します。
  - **checksum** チェックサムを使用してデータベース・ページを検証します。「データベースの妥当性の確認」『SQL Anywhere サーバ - データベース管理』を参照してください。

パーミッション

DBA 権限が必要です。

関連する動作

なし。

備考

このプロシージャは、データベースの各テーブルに対して VALIDATE TABLE 文を呼び出すのと同じです。「VALIDATE 文」 739 ページを参照してください。

tbl_name、*owner_name*、*check_type* の引数の値は、すべて文字列です。値は引用符で囲んで指定します。

プロシージャは Messages という 1 つのカラムを返します。すべてのテーブルが有効である場合、カラムには「**エラーは見つかりませんでした。**」と表示されます。

警告
テーブルまたはデータベース全体の検証は、どの接続においてもデータベースを変更していない場合に実行してください。そうでない場合、実際に破損がなくても、何らかの形でデータベースが破損したことを示す重大な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の式チェックを実行します。

```
CALL sa_validate( owner_name = 'DBA', check_type = 'checksum' );
```

sa_verify_password システム・プロシージャ

現在のユーザのパスワードを検証します。

構文

sa_verify_password(*curr_pwsd* **)**

引数

- ***curr_pwsd*** この CHAR(128) パラメータを使用して、現在のデータベース・ユーザのパスワードを指定します。

備考

このプロシージャは sp_password で使用されます。パスワードが一致した場合、プロシージャは単純に戻ります。一致しなかった場合は、エラー「**ユーザ ID またはパスワードが無効です**」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」 998 ページ

sp_login_environment システム・プロシージャ

ユーザ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です。

このプロシージャ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 [データベース]」 『SQL Anywhere サーバ - データベース管理』

sp_remote_columns システム・プロシージャ

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・プロシージャを使用するには、サーバを 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*** この CHAR(128) パラメータを使用して、CREATE SERVER 文で指定されたサーバ名を含む文字列を指定します。
- ***@table_name*** この CHAR(128) パラメータを使用して、リモート・テーブルの名前を指定します。
- ***@table_owner*** オプションの CHAR(128) パラメータを使用して、*@table_name* の所有者を指定します。
- ***@table_qualifier*** オプションの CHAR(128) パラメータを使用して、*@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	SMALLINT	このフィールドの意味は、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。
scale	SMALLINT	このフィールドの意味は、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、このフィールドは 1 です。それ以外の場合、nullable は 0 です。

備考

CREATE EXISTING 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・データへのアクセス」 『SQL Anywhere サーバ - SQL の使用法』
- 「リモート・データ・アクセスのサーバ・クラス」 『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE SERVER 文」 444 ページ

例

次の例は、Adaptive Server Enterprise サーバ 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

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・プロシージャを使用するには、サーバを 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*** この CHAR(128) パラメータを使用して、プライマリ・テーブルが格納されているサーバを指定します。このパラメータの値は必須です。
- ***@sp_name*** この CHAR(128) パラメータを使用して、プライマリ・キーを格納するテーブルを指定します。このパラメータの値は必須です。
- ***@sp_owner*** オプションの CHAR(128) パラメータを使用して、プライマリ・テーブルの所有者を指定します。
- ***@sp_qualifier*** オプションの CHAR(128) パラメータを使用して、プライマリ・テーブルを格納するデータベースを指定します。

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

このプロシージャは、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・プロシージャは、プライマリ・キーと外部キーの両方のデータベース、所有者、テーブル、カラム、名前と、外部キー・カラムの外部キー・シーケンスを含みます。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」 444 ページ
- 「外部キー」 『SQL Anywhere 10 - 紹介』

例

サーバ asetest で、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取得します。

```
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

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・プロシージャを使用するには、サーバを 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*** オプションの CHAR(128) パラメータを使用して、外部キー・テーブルが格納されているサーバを指定します。このパラメータの値は必須です。
- ***@sp_name*** オプションの CHAR(128) パラメータを使用して、外部キーを格納するテーブルを指定します。このパラメータの値は必須です。

- ***@sp_owner*** オプションの CHAR(128) パラメータを使用して、外部キー・テーブルの所有者を指定します。
- ***@sp_qualifier*** オプションの CHAR(128) パラメータを使用して、外部キー・テーブルを含むデータベースを指定します。

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 の結果セットは、プライマリ・キーと外部キーの両方のデータベース、所有者、テーブル、カラム、名前と、外部キー・カラムの外部キー・シーケンスを含みます。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」 444 ページ
- 「外部キー」 『SQL Anywhere 10 - 紹介』

例

サーバ asetest で、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に関する情報を取得します。

```
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

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*** この CHAR(128) パラメータを使用して、リモート・テーブルが格納されているサーバを指定します。
- ***@table_name*** オプションの CHAR(128) パラメータを使用して、リモート・テーブルを指定します。
- ***@table_owner*** オプションの CHAR(128) パラメータを使用して、リモート・テーブルの所有者を指定します。
- ***@table_qualifier*** オプションの CHAR(128) パラメータを使用して、リモート・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
データベース	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。
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、サーバに指定されたリモート・データ・アクセス・クラスによって、返された情報のカタログの値やデータベースの値は多少異なります。ただし、重要な情報 (カラム名など) は正確です。

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

サーバ上のテーブルのリストを返します。

このシステム・プロシージャを使用するには、サーバを 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*** この CHAR(128) パラメータを使用して、リモート・テーブルが格納されているサーバを指定します。
- ***@table_name*** この CHAR(128) パラメータを使用して、リモート・テーブルを指定します。
- ***@table_owner*** この CHAR(128) パラメータを使用して、リモート・テーブルの所有者を指定します。
- ***@table_qualifier*** CHAR(128) パラメータを使用して、*table_name* が格納されているデータベースを指定します。
- ***@with_table_type*** オプションの BIT パラメータを使用して、リモート・テーブルのタイプを指定します。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
データベース	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。このフィールドの値は、リモート・サーバ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・サーバを設定するときに、特定のサーバ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。このプロシージャは、サーバ上のテーブルのリストを返します。

このプロシージャには 5 つのパラメータを指定できます。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・データへのアクセス」 『SQL Anywhere サーバ - SQL の使用法』
- 「リモート・データ・アクセスのサーバ・クラス」 『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE SERVER 文」 444 ページ

例

サーバ 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 asetest の production 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

リモート・サーバの機能についての情報を表示します。

このシステム・プロシージャを使用するには、サーバを CREATE SERVER 文で定義します。

構文

sp_servercaps(@*sname*)

引数

- ***@sname*** この CHAR(64) パラメータを使用して、CREATE SERVER 文で定義されたサーバを指定します。指定した *@sname* 名は、CREATE SERVER 文で使用するサーバ名と同じでなければなりません。

備考

このプロシージャは、リモート・サーバの機能についての情報を表示します。SQL Anywhere はこの機能の情報を使用して、どのくらいの SQL 文をリモート・サーバに転送できるかを判断します。サーバの機能 (ISYSCAPABILITY と ISYSCAPABILITYNAME) を保持するシステム・テーブルは、SQL Anywhere がリモート・サーバに最初に接続した後でないと、移植されません。

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。

参照

- 「SYSCAPABILITY システム・ビュー」 783 ページ
- 「SYSCAPABILITYNAME システム・ビュー」 784 ページ
- 「リモート・データへのアクセス」 『SQL Anywhere サーバ - SQL の使用法』
- 「リモート・データ・アクセスのサーバ・クラス」 『SQL Anywhere サーバ - SQL の使用法』
- 「CREATE SERVER 文」 444 ページ

例

リモート・サーバ 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

ユーザが 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です。新規接続ごとに、login_procedure で指定されたプロシージャが呼び出されます。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。

このプロシージャは、デフォルトの Sybase 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を作成して、その新しいプロシージャを指すように 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」 971 ページ
- 「login_procedure オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』.

例

sp_tsql_environment プロシージャ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 automatic_TIMESTAMP='On'
  END IF;
  SET TEMPORARY OPTION ansinull='Off';
  SET TEMPORARY OPTION tsql_variables='On';
  SET TEMPORARY OPTION ansi_blanks='On';
  SET TEMPORARY OPTION tsql_hex_constant='On';
  SET TEMPORARY OPTION chained='Off';
  SET TEMPORARY OPTION quoted_identifier='Off';
  SET TEMPORARY OPTION allow_nulls_by_default='Off';
  SET TEMPORARY OPTION float_as_double='On';
  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'
  SET TEMPORARY OPTION close_on_endtrans='Off'
END;
```

xp_cmdshell システム・プロシージャ

プロシージャからシステム・コマンドを実行します。

構文

xp_cmdshell(
command
[, 'no_output'])

引数

- ***command*** この CHAR(8000) パラメータを使用して、システム・コマンドを指定します。
- **'no_output'** オプションの CHAR(254) パラメータを使用して、出力を表示するかどうかを指定します。デフォルトの動作では、出力が表示されます。このパラメータが文字列 **'no_output'** の場合、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。

2 番目のパラメータは、Windows オペレーティング・システムのコンソール・アプリケーションだけに影響します。UNIX の場合は、2 番目のパラメータの設定にかかわらず、出力は表示されません。NetWare の場合は、2 番目のパラメータの設定にかかわらず、実行したすべてのコマンドがサーバ・コンソールに表示されます。

NetWare と Windows CE の場合は、2 番目のパラメータの設定にかかわらず、実行したすべてのコマンドがサーバ・コンソールに表示されます。Windows CE で、プロシージャ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 権限が必要です。

参照

- 「CALL 文」 362 ページ

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。

```
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。

```
xp_cmdshell( 'dir > c:¥¥temp.txt', 'no_output' )
```

xp_read_file システム・プロシージャ

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* **)**

引数

- ***filename*** この LONG VARCHAR パラメータを使用して、内容を返すファイルの名前を指定します。

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。

パーミッション

DBA 権限が必要です。

参照

- 「xp_write_file システム・プロシージャ」 985 ページ
- 「CALL 文」 362 ページ
- 「xp_read_file での openxml の使用」 『SQL Anywhere サーバ - 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

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*** この CHAR(254) パラメータを使用して、入力文字列を指定します。
- ***format*** この CHAR(254) パラメータを使用して、各 *parm* 引数にプレースホルダ (%s) を使用し、入力文字列の形式を指定します。最高で 50 のプレースホルダを *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数にする必要があります。
- ***parm*** これらの CHAR(254) パラメータの中から 1 つ以上のパラメータを使用して、*input_buffer* から抽出したサブ文字列を指定します。最大で 50 のパラメータを指定できます。

備考

xp_scanf システム・プロシージャは、指定した *format* を使用して入力文字列からサブ文字列を抽出し、指定した *parm* 値に結果を入力します。

パーミッション

なし。

参照

- 「CALL 文」 362 ページ

例

次の文は、サブ文字列の Hello と World! を入力バッファの Hello World! から抽出します。文字列は、文字列変数 1 と文字列変数 2 に設定され、その後、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_sprintf システム・プロシージャ

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*** この CHAR(254) パラメータを使用して、結果文字列を格納する出力バッファを指定します。
- ***format*** この CHAR(254)パラメータを使用して、各 *parm* 引数にプレースホルダ (%s) を使用し、結果文字列の形式を指定します。最高で 50 のプレースホルダを *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数にする必要があります。
- ***parm*** 結果文字列で使用する入力文字列があります。最高で 50 の CHAR(254) 引数を指定できます。

備考

xp_sprintf システム・プロシージャは、*format* 引数と *parm* 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」 362 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_write_file システム・プロシージャ

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*** この LONG VARCHAR パラメータを使用して、ファイル名を指定します。
- ***file_contents*** この LONG BINARY パラメータを使用して、ファイルに書き込む内容を指定します。

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 値は、絶対パスまたは相対パスで事前に指定します。*filename* を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・サーバの作業ディレクトリに関連があります。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。

パーミッション

DBA 権限が必要です。

参照

- 「xp_read_file システム・プロシージャ」 982 ページ
- 「CALL 文」 362 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、連絡先の姓 (Surname) と名 (GivenName) を連結した名前が付けられ (たとえば *Reeves_Scott.txt*)、連絡先の住所 (Street)、市 (City)、州 (State) が別々の行に含まれます。

```
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用に画像ファイル (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 が UNIQUE 制約が指定されたローです。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ときに重要です。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、Products.Photo に画像データ (JPG) を格納します。

システム拡張プロシージャ

SQL Anywhere データベースには、一連のシステム拡張プロシージャが含まれています。これらのプロシージャは、dbo ユーザ ID によって所有されています。ユーザがこれらのプロシージャを使用するには、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 と SMTP 用の拡張システム・プロシージャ

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・プロシージャが用意されています。これらのシステム・プロシージャは、拡張システム・プロシージャとして実装されます。各プロシージャは、外部 DLL の関数を呼び出します。

MAPI または SMTP システム・プロシージャを使用するには、データベース・サーバ・コンピュータから 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。

そのためのシステム・プロシージャには、次のものがあります。

- **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。

次のプロシージャは、バックアップが完了したことをユーザ・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
```

次に、メール・システム・プロシージャについて説明します。

xp_startmail システム・プロシージャ

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**　この LONG VARCHAR パラメータを使用して、MAPI ログイン名を指定します。
- **mail_password**　この LONG VARCHAR パラメータを使用して、MAPI パスワードを指定します。

パーミッション

DBA 権限が必要です。

NetWare または 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 リターン・コード」 994 ページを参照してください。

参照

- 「CALL 文」 362 ページ

xp_startsmtp システム・プロシージャ

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*
)

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。
- **smtp_server**　使用する SMTP サーバを指定する LONG VARCHAR パラメータ。これはサーバ名または IP アドレスになります。

- **smtp_port**　SMTP サーバで接続するポート番号を指定するオプションの INTEGER パラメータ。デフォルトは 25。
- **timeout**　データ・サーバからの応答を待機する秒数を指定するオプションの INTEGER パラメータ。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**　このオプションの LONG VARCHAR パラメータは、送信者の電子メールアドレスのエイリアスを指定します。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**　オプションの LONG VARCHAR パラメータは、認証を必要とする SMTP サーバに示すユーザ名を指定します。
- **smtp_auth_password**　オプションの LONG VARCHAR パラメータは、認証を必要とする SMTP サーバに示すユーザ名を指定します。

パーミッション

DBA 権限が必要です。

NetWare ではサポートされません。

備考

xp_startsmtp は、SMTP サーバに接続し、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・プロシージャです。この接続はタイムアウトになります。そのため、executing xp_sendmail を実行する直前に、xp_start_smtp を呼び出すことをおすすめします。

ウィルス・スキャナは xp_startsmtp に影響を与え、その結果エラー・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を大量送信されないようにし、さらに xp_sendmail が正しく実行するの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、大量メール送信の保護を回避する処理を指定できる場合、*dbeng10.exe* と *dbsvr10.exe* を指定します。たとえば、McAfee VirusScan では、この 2 つの処理を大量送信を防ぐために [Properties] 領域の [Excluded Processes] に 2 つの処理を追加できます。

リターン・コード

「SMTP リターン・コード」 995 ページを参照してください。

参照

- 「CALL 文」 362 ページ
- 「xp_startsmtp システム・プロシージャ」 988 ページ
- 「xp_stopsmtp システム・プロシージャ」 994 ページ

xp_sendmail システム・プロシージャ

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** = *file-name*]
[, **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ており、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。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します。
- **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。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。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します。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。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します。デフォルト値は NULL です。
- **query** この LONG VARCHAR は Transact-SQL で使用されます。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。デフォルト値は NULL です。"message" は予約語であるため、"message" パラメータ名は二重引用符で囲む必要があります。「予約語」 4 ページを参照してください。
- **attachname** この LONG VARCHAR パラメータは Transact-SQL で使用されます。デフォルト値は NULL です。
- **attach_result** この INT パラメータは Transact-SQL で使用されます。デフォルトは 0 です。
- **echo_error** この INT パラメータは Transact-SQL で使用されます。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。デフォルト値は NULL です。
- **no_column_header**　この INT パラメータは Transact-SQL で使用されます。デフォルトは 0 です。
- **'no_output'**　この INT パラメータは Transact-SQL で使用されます。デフォルトは 0 です。
- **width**　この INT パラメータは Transact-SQL で使用されます。デフォルトは 80 です。
- **separator**　この CHAR(1) パラメータは Transact-SQL で使用されます。デフォルトは 9 です。
- **dbuser**　この LONG VARCHAR パラメータは Transact-SQL で使用されます。デフォルトは guest です。
- **dbname**　この LONG VARCHAR パラメータは Transact-SQL で使用されます。デフォルトは master です。
- **type**　この LONG VARCHAR パラメータは Transact-SQL で使用されます。デフォルト値は NULL です。
- **include_query**　この INT パラメータは Transact-SQL で使用されます。デフォルトは 0 です。
- **content_type**　この LONG VARCHAR パラメータは、「message」パラメータのコンテンツ・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 の値が検証され、無効なコンテンツ・タイプを設定すると、無効または一貫性がない電子メールが送信されます。

パーミッション

DBA 権限が必要です。

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 パラメータはサポートされません。

NetWare で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・プロシージャです。このプロシージャには、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 email. xp_sendmail の要件を理解するユーザが、content_type として ASIS を受け入れる場合に使用します。content_type が ASIS に設定されると、xp_sendmail はメッセージ・ボディ ("message") が正しく構成されたヘッダ付き電子メールであると仮定し、新規のヘッダは追加しません。ASIS を指定して、複数のコンテンツ・タイプを含む複数のメッセージを送信できます。MIME の詳細については、RFC 2045 ー 2049 (http://www.ietf.org/) を参照してください。

include_file パラメータで指定された添付ファイルは、application/octet-stream MIME タイプとして base64 エンコーディングで送信され、データベース・サーバに作成されます。

リターン・コード

「MAPI と SMTP 用のシステム・プロシージャのリターン・コード」 994 ページを参照してください。

参照

- 「CALL 文」 362 ページ
- 「xp_startmail システム・プロシージャ」 987 ページ
- 「xp_startsmtp システム・プロシージャ」 988 ページ
- 「xp_stopmail システム・プロシージャ」 993 ページ
- 「xp_stopsmtp システム・プロシージャ」 994 ページ

例

次の呼び出しは、メール添付書類としてファイル *prices.doc* を持つメッセージを、ユーザ 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、多様な xp_sendmail システム・プロシージャの使用例を示します。

```
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 yourself 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_stopmail システム・プロシージャ

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 権限が必要です。

NetWare または 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・プロシージャです。

リターン・コード

「MAPI リターン・コード」 994 ページを参照してください。

参照

- 「CALL 文」 362 ページ

xp_stopsmtp システム・プロシージャ

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 権限が必要です。

NetWare ではサポートされません。

備考

xp_stopsmtp は、電子メール・セッションを終了するシステム・プロシージャです。

リターン・コード

「SMTP リターン・コード」 995 ページを参照してください。

参照

- 「CALL 文」 362 ページ

MAPI と SMTP 用のシステム・プロシージャのリターン・コード

MAPI と SMTP 用の拡張システム・プロシージャは、次のリターン・コードを使用します。

MAPI リターン・コード

リターン・コード	意味
0	成功
2	xp_startmail 失敗
3	xp_stopmail 失敗
5	xp_sendmail 失敗
11	受信者不明確
12	添付書類紛失
13	ディスクが満杯

リターン・コード	意味
14	失敗
15	メモリ不足
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
20	認識できない受信者
21	ログイン失敗
22	セッション数過多
23	ユーザ・アボート
24	MAPI がない
25	startmail がない

SMTP リターン・コード

リターン・コード	意味
0	成功
100	ソケット・エラー
101	ソケットのタイムアウト
102	SMTP サーバのホスト名を解決できない
103	SMTP サーバに接続できない
104	サーバ・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が SMTP ではない場合など)
421	<*domain*> サービスを使用できない。送信チャネルを閉じる
450	要求されたメール・アクションが実行されない。メールボックスを使用できない
451	要求されたアクションが実行されない。ローカルの処理エラー
452	要求されたアクションが実行されない。システムの記憶領域不足

リターン・コード	意味
500	構文エラー。コマンドが認識されない(コマンドが長すぎるなどのエラーを含む場合がある)
501	パラメータまたは引数の構文エラー
502	コマンドが実装されていない
503	コマンドの順序が不正
504	コマンド・パラメータが実装されていない
550	要求されたアクションが実行されない。メールボックスを使用できない(たとえば、メールボックスが見つからない場合、アクセス権がない場合、リレーが許可されていない場合など)
551	ユーザがローカルでない。<*forward-path*> を試すこと
552	要求メール・アクションがアボートされた。記憶領域の割り付けを超えた
553	要求されたアクションが実行されない。メールボックス名が許可されない(たとえば、メールボックスの構文が正しくない場合など)
554	トランザクションが失敗した

その他の拡張システム・プロシージャ

その他のシステム拡張システム・プロシージャを使用して、システム・コマンドを実行できます。たとえば、ファイル I/O や文字列操作にこれらのシステム・プロシージャを使用できます。

xp_msver システム・プロシージャ

データベース・サーバ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(*string* **)**

- ***string*** 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で囲みます。

引数	説明
ProductName	製品 (Sybase 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： 10.0.1 (30)

引数	説明
CompanyName	次の文字列を返す： iAnywhere Solutions, Inc.
FileDescription	製品名、オペレーティング・システム名の順に返す
LegalCopyright	ソフトウェアの著作権を示す文字列を返す
LegalTrademarks	ソフトウェアの商標を示す文字列を返す

備考

xp_msver は、製品、会社、バージョンなどの情報を返します。

パーミッション

なし。

参照

- 「システム関数」 100 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	Description
9.0.0 (1912)	Sybase SQL Anywhere Windows NT

Adaptive Server Enterprise のシステム・プロシージャとカタログ・プロシージャ

Adaptive Server Enterprise には、システム・プロシージャとカタログ・プロシージャがあります。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・プロシージャです。

SQL Anywhere では、このプロシージャの一部がサポートされています。ただし、このプロシージャの使用方法の詳細については、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

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・プロシージャについて説明します。

これらのプロシージャは Adaptive Server Enterprise やバージョン 12 より前の Adaptive Server IQ 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、プロシージャを調べてください。ストアド・プロシージャのテキストを参照するには、Sybase Central でプロシージャを開くか、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 で全テキストが表示されない場合は、[コマンド] – [オプション] を選択し、新しい [表示カラムの制限値] を入力して出力の幅をリセットします。

システム・プロシージャ名	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・プロシージャの PRINT と RAISERROR の呼び出しで使用する場合 ISYSUSERMESSAGE へのユーザ定義メッセージを追加する
sp_addtype	ユーザ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 ID を追加する
sp_changegroup	ユーザのグループを変更、またはグループにユーザを追加する
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	ユーザ定義メッセージを削除する

システム・プロシージャ名	説明
sp_droptype	ユーザ定義データ型を削除する
sp_dropuser	データベースからユーザ ID を削除する
sp_getmessage	PRINT 文と RAISERROR 文の場合 ISYSUSERMESSAGE から保存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、トリガ、またはビューのテキストを表示する
sp_password	ユーザ 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

SQL Anywhere は、Adaptive Server Enterprise カタログ・プロシージャのサブセットを実装します。次に、実装されたカタログ・プロシージャを示します。

カタログ・プロシージャ名	説明
sp_column_privileges	サポート対象外
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・プロシージャの入力パラメータとリターン・パラメータ情報を返す
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

索引

記号

A

B

C

D

E

F

G

H

I

J

K

L

M

P

Q

R

S

T

U

V

W

X

Y

Z

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

へ

ほ

ま

め

も

ゆ

よ

ら

り

る

れ

ろ

わ

