DBMaster

SQL 文と関数参照編

SYSCOM

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive
Santa Clara, CA 95050, U.S.A.

Contact Information:
CASEMaker US Division
E-mail : info@casemaker.com
Europe Division
E-mail : casemaker.europe@casemaker.com
Asia Division
E-mail : casemaker.asia@casemaker.com(Taiwan)
E-mail : info@casemaker.co.jp(Japan)

www.casemaker.com
www.casemaker.com/support

Document No. 645049-235206/DBM53J-M08312012-SQLR
発行日:2012-08-31

本文には記されていない新しい機能についての説明は、CASEMakerのDBMasterをインストールしてからREADME.TXTを読んでください。

登録商標

注意事項

本書で記述されるソフトウェアは、ソフトウェアと共に提供される使用許諾書に基づきます。

保証については、ご利用の販売店にお問い合わせ下さい。販売店は、特定用途への本コンピュータ製品の商品性や適合性について、代表または保証しません。販売店は、突然の衝撃、過度の熱、冷気、湿度等の外的な要因による本コンピュータ製品へ生じたいかなる損害に対しても責任を負いません。不正な電圧や不適合なハードウェアやソフトウェアによってもたらされた損失や損害も同様です。

本書の記載情報は、その内容について十分精査していますが、その誤りについて責任を負うものではありません。本書は、事前の通知無く変更することがあります。

目次

1 はじめに

SQL文と関数参照編にようこそ。DBMasterは、強力で柔軟なSQL*データベース管理システム*(*DBMS*)であり、会話型の構造的問い合わせ言語(SQL)、Microsoftのオープン・データベース結合(ODBC)互換インタフェース、C言語の*埋め込み*SQL(*ESQL*/C)をサポートします。ODBCインタフェースは唯一のオープン・アーキテクチャであり、多種多様なプログラミングツールによる顧客アプリケーションの構築、既存のODBC適合アプリケーションによるデータベース問い合わせを可能にします。

DBMasterは、シングル・ユーザーの個人データベースから企業全体に分散するデータベースまで容易にスケール化することができます。DBMasterの先進的なセキュリティ、整合性、信頼性の機能は、どのようなデータベース構成でも、重要データの安全性を確実に保証します。広範なクロス - プラットフォームをサポートし、現在あるハードウェアの効力を高め、需要の増大に応じてより強力なハードウェアに拡大し、グレードアップすることを可能にします。

DBMasterは、優れたマルチメディア処理機能をもち、あらゆるタイプのマルチメディアデータの格納、探索、検索、操作を可能にします。バイナリラージオブジェクト(BLOB)は、DBMasterの先進的セキュリティ・損傷リカバリ機構を全面的に利用して、マルチメディアデータの整合性を確実にします。ファイルオブジェクト(FO)は、元のアプリケーションで各ファイルを編集する機能を維持しつつ、マルチメディアデータを管理します。

1.1 その他のマニュアル

DBMasterには、本マニュアル以外にも多くのユーザーガイドや参照編があります。特定のテーマについての詳細は、以下の書籍をご覧下さい。

- DBMasterの能力と機能性についての概要は、*「DBMaster入門編」*をご覧下さい。
- DBMasterの設計、管理、保守についての詳細は、*「データベース管理者参照編」*をご覧下さい。
- DBMasterの管理についての詳細は、*「JServer Managerユーザーガイド」*をご覧下さい。
- DBMasterの環境設定についての詳細は、*「JConfiguration Tool参照編」*をご覧下さい。
- DBMasterの機能についての詳細は、*「JDBA Toolユーザーガイド」*をご覧下さい。
- DBMasterで使用しているdmSQLのインターフェースについての詳細は、*「dmSQLユーザーガイド」*をご覧下さい。
- ESQLプログラムについての詳細は、*「ESQL/Cプログラマー参照編」*をご覧下さい。
- ODBCとJDBCプログラムについての詳細は、*「ODBCプログラマー参照編」*と*「JDBCプログラマー参照編」*をご覧下さい。
- エラーと警告メッセージについての詳細は、*「エラー・メッセージ参照編」*をご覧下さい。
- ネイティブDCI APIについての詳細は、*「DCIユーザーガイド」*をご覧下さい。
- SQLストアドプロシージャについての詳細は、*「SQLストアドプロシージャ参照編」*をご覧下さい。

1.2 字体の規則

本書は、標準の字体規則を使用しているので、簡単かつ明確に読むことができます。

斜体	斜体は、ユーザー名や表名のような特定の情報を表します。斜体の文字そのものを入力せず、実際に使用する名前をそこに置き換えてください。斜体は、新しく登場した用語や文字を強調する場合にも使用します。
太字	太字は、ファイル名、データベース名、表名、カラム名、関数名やその他同様なケースに使用します。操作の手順においてメニューのコマンドを強調する場合にも、使用します。
キーワード	文中で使用するSQL言語のキーワードは、すべて英大文字で表現します。
小さい英大文字	小さい英大文字は、キーボードのキーを示します。２つのキー間のプラス記号 (+) は、最初のキーを押したまま次のキーを押すことを示します。キーの間のコンマ(,)は、最初のキーを放してから次のキーを押すことを示します。
注	重要な情報を意味します。
➲ プロシージャ	一連の手順や連続的な事項を表します。ほとんどの作業は、この書式で解説されます。ユーザーが行う論理的な処理の順序です。
➲ 例	解説をよりわかりやすくするために与えられる例です。一般的に画面に表示されるテキストと共に表示されます。
コマンドライン	画面に表示されるテキストを意味します。この書式は、一般的にdmSQLコマンドやdmconfig.iniファイルの内容の入/出力を表示します。

2 SQLの基礎

本書は、DBMasterのSQL言語を使用する読者を想定しています。読者には、dmSQLコマンドラインユーティリティを使用してデータベース問合せを行うユーザーから、ESQL/CとODBC適合インタフェースを使用して顧客アプリケーションを開発するプログラマーまでの全ての人が含まれます。

本書はDBMasterのSQL言語の完全な参照情報を提供し、各SQL文の構文を記載します。使用例と図を使い明確に理解できるようにします。

2.1 構文ダイアグラム

SQLの構文は構文ダイアグラムで示します。構文ダイアグラムは、SQL文の作成時に構文オプションを調べるときに役立ちます。構文ダイアグラムの例を次に図示します。構文ダイアグラムは、始点から終点までの線をたどって使用します。進路上にあるSQL文の要素は必須のものです。進路から分岐した要素は、追加オプションや構文の柔軟性を示します。

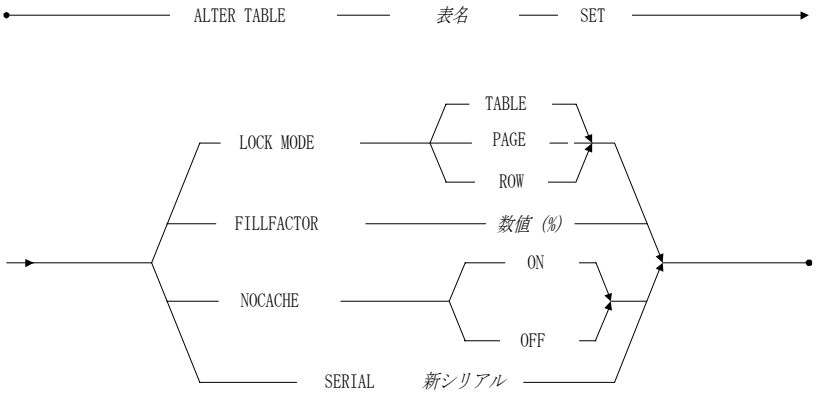


図2-1：構文ダイアグラムの簡単な例

*斜体*の字句は、データベースで使われている実際の名前を指定する場所を示します。この部分は実際の名前で置き換えます。上のダイアグラムでは、*表名*をデータベースに実際ある表の名前に置き換えます。例えば、**Tutorial**という名前のデータベースで*表名*を**Customers**に置き換えると、**Customers**表にSQL文を実行します。

矢印の向きにも注意する必要があります。SQL文の中で項目リストを与えることができる場所では、構文ダイアグラムの矢印は、項目の循環パスになるように付けられます。ダイアグラムの矢印をたどると、カラム名をカンマで分離したカラム名リストの循環パスになります。

2.2 データ型

表のカラムを定義するときは、カラムのデータ型を指定します。適切なデータ型を選ぶためには、各フィールドがどのように使用されるかを理解しなければなりません。不適切なデータ型を選ぶと、データベースのスペースを浪費し、アプリケーション・プログラムで使用可能な形式にデータを変換する余分なステップが必要になります。DBMasterは、以下のデータ型をサポートします：

BIGINT 、 BINARY(size) 、 BIGSERIAL 、 CHAR(size) 、 NCHAR(size) 、 DATE 、 DECIMAL （ NUMERIC ） 、 DOUBLE 、 FILE 、 FLOAT 、 INTEGER 、 LONG VARBINARY （ BLOB ） 、 LONG VARCHAR （ CLOB ） 、 OID 、 SERIAL 、 SMALLINT 、 TIME 、 TIMESTAMP 、 VARCHAR、NVARCHAR(size)、メディアタイプ。

BIGINT

BIGINTは、19桁の精度とスケール0をもつ符号付き真数のデータ型です。BIGINTデータ型は8バイトのストレージを使用し、最大値は+9 223 372 036 854 775 807、最小値は−9 223 372 036 854 775 808です。

許容範囲外の値をBIGINT、INTEGERなどのデータ型から移そうとすると、変換エラーが表示されデータは移せません。

➲ **例** 1

```
37654
```

➲ **例** 2

```
857823
```

BIGSERIAL (開始番号)

BIGSERIALは、連続するシリアル番号を生成する特殊なデータ型です。データベースの各表には整数番号が一つ割り当てられており、この整数番号

を使用して表に一意なシリアル番号が生成されます。DBMasterは各表の整数番号を内部的に管理・維持し、使用される度に自動的に1増分します。

BIGSERIALカラムを定義するときに、開始番号のオプションパラメタを指定してシリアル番号の初期値を設定することができます。開始番号を指定しないと初期設定値の1になります。表は1個だけBIGSERIALデータ型のカラムをもつことができます。

BIGSERIAL番号は整数値であり。BIGSERIALは、8バイトのストレージを占め、精度19とスケール0をもつ符号付き真数のデータ型です。BIGSERIAL 最大値 9,223,372,036,854,775,806 と 最小値 -9,223,372,036,854,775,808をもちます。

BIGSERIALカラムに次のシリアル番号を挿入するには、新しい行を挿入するときにBIGSERIALカラムをNULLにします。表の内部シリアル番号が新レコードのBIGSERIALカラムに挿入され、自動的に1増分されます。

NULLの代わりに整数値を与えたBIGSERIALカラムをもつ行を挿入すると、次のシリアル番号の代わりに与えられた整数値が挿入されます。内部シリアル番号は増分されません。与えられた値が最後に生成したシリアル番号よりも大きいときは、内部シリアル番号が与えられた値にリセットされます。

➲ 例 1

```
10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007
```

➲ 例 2

```
10000, 10001, 5000, 10002, 10003, 11000, 11001, 11002
```

BINARY(サイズ)

BINARYは、固定長の任意のバイナリの値をもつことができるデータ型です。BINARYカラムの最小サイズは1バイト、最大サイズは3992バイトです。BINARYカラムを作成するときは、サイズパラメタの値を指定します。BINARYカラムにカラムサイズよりも短いデータを入力すると、0-値のバイトが補充されます。

文字データは、CHARデータ型と同様、文字データを引用符(‘ ’)で括って入力します。BINARYカラムの文字データは、入力された文字ではなく、文字のASCIIコードを表す16進数として格納されます。

直接16進数を入力するには、16進文字列を引用符で括り後ろに16進の値であることを示す‘x’を付けます(‘ ’x)。各バイトの値を16進数で表すには2桁が必要です。偶数桁の16進数字を入力します。

➲ 例1

```
‘AaBbCcDdEe’x
```

➲ 例2

```
‘41614262436344644565’x
```

CHAR(サイズ)

CHARは、キーボード上の任意の文字をもつことができる固定長のデータ型です。CHARカラムの最小サイズは1バイト、最大サイズは3992バイトです。CHARカラムの作成時には、サイズパラメタの値を指定します。

CHARカラムにカラムサイズよりも短いデータを入力すると、スペースが補充されます。CHARデータを入力するときは、文字データを引用符(‘ ’)で括ります。全角文字は2バイトを使用します。全角文字を入力するカラムのサイズを指定するときは、この点を考慮に入れます。

➲ 例

```
‘This is a CHAR string.’
‘This is another CHAR string.’
```

DATE

DATEは、日付(年、月、日)をもつ固定長のデータ型です。DATE型は4バイトのストレージを使用します。年の有効値は 0001～9999です。

DATEデータ型には、多くの入力と出力のフォーマットがあります。データベース上の日付が正しく表示されなかったり、正しい日付なのに入力することができない場合は、日付の入力フォーマットと出力フォーマットが正しいかを確かめてください。

➲ 例

```
'0001/01/01'
'1999/12/31'
```

DECIMAL (NUMERIC)

DECIMALは、可変長の精度とスケールをもつ符号付き真数のデータ型です。精度は小数点以上と以下の両方の合計桁数を表し、スケールは小数点以下の桁数を表します。精度の初期設定値は17、最大値は38です。スケールの初期設定値は6です。

DECIMALカラムのストレージサイズは、実際に入力した数値で決められます。カラム定義で指定した精度とスケール、あるいは初期設定の精度とスケールではありません。

ストレージサイズは次の式で計算します：

$$\text{バイト数} = \frac{p+2}{2} + 2$$

例えば、9283.83は6バイトに格納されます。

このバイト数の計算を次ページに示します：

$$\text{バイト数} = \frac{p+2}{2} + 2$$
$$= \frac{6+2}{2} + 2$$
$$= 6$$

許容範囲外の値をFLOATやDOUBLEなどのデータ型から移そうとすると、変換エラーが表示されデータは移せません。DECIMALデータ型はDECと略記することができます。

➲ 例

```
3452.8373645
736.383732652
```

DOUBLE

DOUBLEは、15桁の精度をもつ符号付き概数のデータ型です。精度は小数点以上と以下の両方の合計桁数を表します。DOUBLEは8バイトのストレージを使用し、有効な入力値の範囲は1.0E308～–1.0E308です。

最小の有効値は1.0E-308および–1.0E-308です。

➲ 例

```
2.89837457884451E285
-1.93873634847372E-174
```

FILE

FILEは48バイトのストレージを占める構造的データ型であり、任意の既存ファイルを外部ファイルとして格納します。他のデータと同様にDBMasterで参照することができます。FILEはLONG VARCHARやLONG VARBINARYデータ型に似ていますが、データは、内部オブジェクトとしてではなく、外部ファイルとして格納されます。ファイルを使用する第三者ツールは、元のフォーマットでデータにアクセスし変更することができるだけでなく、データベースに変更を登録するためにデータを再インポートする必要も無くなります。ファイルオブジェクトのパス名の最大長は255字です。

FILEカラムには、システムカタログ表のレコードへの参照が格納されます。システムカタログは、データベース上のファイルオブジェクトを見つけるための情報をもっています。FILEカラムを表示しても、カラムに格納

されている情報は表示されません。DBMasterは、ファイル名、ファイルサイズ、ファイル内容の3種類のビューのどれかで、システムカタログやファイルに格納されている情報を表示します。

FILEデータ型は、システムファイルオブジェクトまたはユーザーファイルオブジェクトのいずれかでデータを格納することができます。システムファイルオブジェクトは、データベースのファイルオブジェクトディレクトリに元のファイルをコピーして固有の名前を付けます。コピーされたファイルはデータベースによって管理され、FILEカラムからの参照が無くなると削除されます。ユーザーファイルオブジェクトは、元のファイルの場所と名前をそのままにし、ファイルへのリンクを作成します。ファイルはユーザーが作成したものであり、データベースからの参照が無くなっても削除されません。ユーザーファイルオブジェクトとしてファイルを挿入するときは、ファイルの読み込み権限をDBMasterに与えておきます。

複数のレコードが同じファイルを参照しても、1個だけコピーを格納してレコード間で共有し、ディスクを節約します。しかしユーザーからは、常にレコード毎に固有のファイルがあるように見えます。共有ファイルを更新するときは、新しいファイルがDBMasterによって透過的に生成され、他のレコードが共有しているファイルは変更されません。他のユーザーは依然として元のファイルを見ることになります。こうして、あるレコードのファイル変更が他のレコードに影響しないようにしています。

FLOAT

FLOATは、15桁の精度をもつ符号付き概数のデータ型です。精度は小数点以上と以下の両方の合計桁数を表します。FLOATは8バイトのストレージを使用し、有効な入力値の範囲は1.0E308～–1.0E308です。デフォルトのFLOAT型はキーワードDB_FLTDBの設定でREALまたはDOUBLEに定義できます。

最小の有効値は**1.0E-308** および**–1.0E-308**です。

➲ **例** 1

```
2.89837457884451E285
```

➲ 例 2

```
-1.93873634847372E-174
```

INTEGER

INTEGERは、10桁の精度とスケール0をもつ符号付き真数のデータ型です。INTEGERデータ型は4バイトのストレージを使用し、最大値は2,147,483,647、最小値は-2,147,483,648です。

許容範囲外の値をDOUBLEなどのデータ型から移そうとすると、変換エラーが表示されデータは移せません。INTEGERデータ型はINTと略記することができます。

➲ 例

```
393848
-298376
```

LONG VARBINARY(BLOB)

BLOBは、可変長の任意のバイナリの値をもつことができるデータ型です。BLOBカラムの最大長は約8TB です。入力データだけがデータベースに格納され、BINARYのように0値が補充されることはありません。

文字データは、CHARデータ型のように引用符(‘ ’)で括って入力します。BLOBカラムは、実際に入力された文字ではなく、文字のASCIIコードを表す16進数としてデータを格納します。

直接16進数を入力するには、16進文字列を引用符で括り後ろに16進の値であることを示す‘x’を付けます(‘ ’x)。各バイトの値は2桁の16進数で表します。偶数桁の16進数字の値を入力します。

➲ 例

```
‘AaBbCcDdEe’
‘41614262436344644565’x
```

LONG VARCHAR(CLOB)

CLOBは、可変長の任意の文字をもつことができるデータ型です。CLOBカラムの最大長は、約8TB です。

スペースを補充するCHARデータ型と異なり、入力したデータだけがデータベースに格納されます。CLOBカラムに文字データを入力するときは、文字データを引用符(‘ ’)で括ります。全角文字は2バイトを使用します。

➲ **例**

```
‘This is a LONG VARCHAR string.’
‘This is another LONG VARCHAR string.’
```

NCHAR(size)

NCHARデータ・タイプはどんなUnicode文字も含むことができる固定長データ型です。各Unicode文字は、UTF16リトル・エンディアン(LE)符号づけの中で記憶装置を2バイト占めます。(サイズ)パラメーターは、カラムの2バイトの文字の数を決定します。NCHARカラムを作る場合(サイズ)パラメーターを入力しなければならず、範囲は最小1から最大1996です。NCHARデータが、カラム長より短いカラムに入力される場合、データがスペース文字で埋められるでしょう。NCHARデータを入力する場合、シングルクォートでUnicode文字を囲んで、「N」を前に付けてください。

➲ **例** 1

下記は、Unicodeデータ・エントリーのシンタックスを実証します:

```
N'Unicode Data'
```

NCHARデータが16進法のフォーマットに入力される場合は、引用符(‘ ’)で16進法のストリングを囲んで、「u」文字を追加してください。

➲ **例** 2

下記は、3文字の16進法Unicodeデータ・エントリーのシンタックを実証します:

```
'610a620b63f1'u
```

文字ストリングがUnicodeカラムへ入力された時「N」が前に付けられない場合は、自動的にローカル・コードからUnicodeに変換されるでしょう。Unicode文字が規則的なCHARタイプ・カラムに入力される場合、Unicode文字はdmconfig.iniパラメーター**Db_LCode**によって定義されたローカル・コードに変換されるでしょう。ローカル・コードに定義されない文字は□にによって表わされるでしょう。

NCHARデータ型の同意語はNATIONAL CHAR(サイズ)およびNATIONAL CHARACTER(サイズ)です。

NVARCHAR(size)

NVARCHARデータ・タイプはどんなUnicode文字も含むことができる可変長さのデータ・タイプです。各Unicode文字は、UTF16リトル・エンディアン(LE)符号づけの中で記憶装置を2バイト占めます。(サイズ)パラメーターは、カラムの2バイトの文字の数を決定します。NVARCHARカラムを作る場合(サイズ)パラメーターを入力しなければならず、範囲は最小1から最大1996までです。

NVARCHARデータが、カラム長さより短いカラムに入力される場合、データがスペースで当てがわれません。NVARCHARデータを入力する場合、シングルクォートでUnicode文字を囲んで、「N」を備えた引用の前に付けてください。

➲ **例** 1

下記は、Unicodeデータ・エントリーのシンタックスを実証します:

```
N'Unicode Data'
```

NVARCHARデータが16進法のフォーマットに入力される場合は、引用で16進法のストリングを囲んで、「u」文字を追加してください。

➲ **例** 2

下記は、3文字の16進法Unicodeデータ・エントリーのシンタックスを実証します:

```
'610a620b63f1'u
```

文字ストリングがUnicodeカラムへ入力された時「N」が前に付けられない場合は、自動的にローカル・コードからUnicodeに変換されるでしょう。Unicode文字が規則的なVARCHARタイプ・カラムに入力される場合、Unicode文字はdmconfig.iniパラメーター**Db_LCode**によって定義されたローカル・コードに変換されるでしょう。ローカル・コードに定義されない文字は□によって表わされるでしょう。

NVARCHARデータ・タイプの同意語はNATIONAL CHAR VARYING(サイズ)、NCHAR VARYING(サイズ)、NATIONAL VARCHAR(サイズ)およびNATIONAL CHARACTER VARYING(サイズ)です。

OID

OID（オブジェクトID）は、データベースに格納されるオブジェクト（レコード、BLOB）に割り当てられる一意IDのデータ型です。OIDは16バイトのストレージを使用し、精度10とスケール0をもちます。OIDは自動的に生成され、各レコードに挿入されます。OIDはDBMasterによって内部的に管理・維持され、ユーザーが直接使用することはありません。

OIDの値は、オブジェクトのストレージ場所に関係します。連続して生成された二つのOIDが順序通り並んでいる保証はありません。

OIDは表の隠しカラムです。「SELECT * FROM CUSTOMERS:」のような問合せでは表示されません。OIDカラムを検索するには、問合せの中でカラム名として‘OID’を明示的に指定します。

問合せの中でOIDを使用して表からデータを検索し、次にOIDを使用して表のデータを更新することは可能ですが、SQL言語では、このような使用はあまり行いません。OIDは、通常、プログラミングの内部インタフェースに使用します。会話型のdmSQL環境で直接使用することはありません。

REAL

REAL は、7桁の精度をもつ符号付き概数のデータ型です。精度は小数点以上と以下の両方の合計桁数を表します。REALは4バイトのストレージを

使用し、有効な入力値の範囲は3.402823466E38 ～ –3.402823466E38です。最小の有効値は1.175494351E-38および–1.175494351E-38です。DOUBLEのようなデータ型から最大値を超える値を含む移動は失敗となり、DBMasterは変換エラーを返します。

➲ 例 1

```
3.583837E34
```

➲ 例 2

```
-1.873653E-21
```

SERIAL(開始番号)

SERIALは、連続するシリアル番号を生成する特殊なデータ型です。データベースの各表には整数番号が一つ割り当てられており、この整数番号を使用して表に一意なシリアル番号が生成されます。DBMasterは各表の整数番号を内部的に管理・維持し、使用される度に自動的に1増分します。SERIALカラムを定義するときに、開始番号のオプションパラメタを指定してシリアル番号の初期値を設定することができます。開始番号を指定しないと初期設定値の1になります。表は1個だけSERIALデータ型のカラムをもつことができます。

シリアル番号は整数値であり、SERIALデータ型はINTEGERデータ型と同じ属性をもちます。SERIALは、4バイトのストレージを占め、精度10とスケール0をもつ符号付き真数のデータ型です。SERIALはINTEGERと同じ値の範囲をもち、最大値2,147,483,646と最小値–2,147,483,648をもちます。

SERIALカラムに次のシリアル番号を挿入するには、新しいレコードを挿入するときにSERIALカラムをNULLにします。表の内部シリアル番号が新レコードのSERIALカラムに挿入され、自動的に1増分されます。

NULLの代わりに整数値を与えたSERIALカラムをもつ行を挿入すると、次のシリアル番号の代わりに与えられた整数値が挿入されます。内部シリアル番号は増分されません。与えられた値が最後に生成したシリアル番号よりも大きいときは、内部シリアル番号が与えられた値にリセットされます。

➲ 例

```
100, 101, 102, 103, 104, 105, 106, 107
100, 101, 50, 102, 103, 110, 111, 112
```

SMALLINT

SMALLINTは、精度5とスケール0をもつ符号付き真数のデータ型です。SMALLINTは2バイトのストレージを使用し、最大値32,767と最小値-32,768をもちます。

許容範囲以外の値をINTEGERやDOUBLEなどのデータ型から移そうとすると、変換エラーが表示されデータは移されません。

➲ 例

```
4769
8376
```

TIME

TIME型データにはTIME literalとTIME constantの2種類があります。TIME constant は時間の固定モーメントです。どちらのTIME型も固定長で、4バイトのストレージを使用します。オプションの‘AM’あるいは‘PM’を指定しないと、時刻は初期設定の24時間形式で挿入されます。

TIMEデータ型には、多くの入力フォーマットと出力フォーマットがあります。データベース上の時刻が正しく表示されなかったり、正しい時刻なのに入力することができない場合は、時刻の入力フォーマットと出力フォーマットが正しいかチェックして確かめてください。

➲ 例

```
‘22:04:05’
‘22:04:05’t
TIME ‘22:04:05’
‘10:04:05.666 PM’
10:04:05 PM’t
```

```
TIME 10:04:05 PM'
```

TIMESTAMP

TIMESTAMPは、日付と時刻をもつ固定長のデータ型です。11バイトのストレージを使用し、精度17とスケール10をもちます。年の有効値は0001～9999です。オプションの'AM'あるいは'PM'を指定しないと、時刻は初期設定の24時間形式で挿入されます。

TIMESTAMPは、TIMEとDATEの入力フォーマットと出力フォーマットを使用して値を表示し、入力値の正しさを検査します。データベース上の値が正しく表示されなかったり、正しい値なのに入力することができない場合は、入力フォーマットと出力フォーマットが正しいか確かめてください。

➲ 例

```
'1997/01/01 10:02:03.444 PM'
'1997/01/01 22:02:03'ts
TIMESTAMP '1997/01/01 10:02:03'
'01.01.1997 22:02:03'
'01.01.1997 22:02:03'ts
TIMESTAMP '01.01.1997 22:02:03'
```

VARCHAR(サイズ)

VARCHARは、任意の文字をもつことができる可変長のデータ型です。VARCHARカラムの最小サイズは1バイト、最大サイズは3992バイトです。VARCHARカラムを作成するときはカラムの最大サイズを指定します。

VARCHARカラムには、入力された文字だけが格納されます。データを入力するには、文字データを引用符(' ')で括ります。全角文字は2バイトを使用します。全角文字を入力するカラムのサイズを指定するときは、この点を考慮に入れます。

➲ 例

```
'This is a VARCHAR string.'
'This is another VARCHAR string.'
```

Media Types

Microsoft™ Word™ドキュメント用全文検索のようなメディア・プロセス機能の補完するために、ラージオブジェクト・カラムをメディア・タイプとして指定することができます。次のメディア・タイプが利用可能です :MsWordType 、 HtmlType 、 XmlType 、 MsPPTType 、 MsExcelType 、 PDFType 、 MsWordFileType 、 HtmlFileType 、 XmlFileType 、 MsPPTFileType、 MsExcelFileType、PDFFileType。

メディア・タイプは既存のデータ・タイプのドメインとして扱われます。MsWordType、MsPPTType、MsExcelType、PDFType、HtmlTypeとXmlTypeは LONG VARBINARY に相当します。また、 MsWordFileType 、 HtmlFileType 、 MsPPTFileType 、 MsExcelFileType と PDFFileType および XmlFileTypeはFILEタイプ・カラムに相当します。カラムのデータ・タイプを別のものに変更するALTER TABLEを使用する場合に重要となります。各メディア型の特性はものとのデータ型の特性と類似しています。

XMLTYPEの特性:

- 整形式のXML文書チェック: INSERT/UPDATEされるXML文書は整形式でなければなりません。
- XML妥当性認証:XML型のカラムを作成するときUDFの妥当性認証をオプションとして設定、DBMasterは設定を使用してXMLの妥当性を認証します。
- XMLデータが 原型のフォーマットで保存されます。
- クエリによるXPath サーチ: オプションでXPathを設定、Extract関数をXMLデータ内のQUERY/LOCATEノードに使用する。
- XPathによるUpdate XML文書の設定。

- XPath extractに索引構築:頻繁に使うXPATHクエリ式に索引を構築しXPATHクエリを高速化。
- XML型カラムまたはデータ型をXML型への変更不可。

2.3 予約語

以下のキーワードの一覧は、識別子として使用されます。以下の予約語がキーワードとして使用され、希望したSQL文が実行されない場合は、エラー・メッサージERR_RESERVED_WORDが戻ります。

ABSOLUTE | ACTION | ADD | ADMIN | AFTER | AGGREGATE | ALIAS | ALLOCATE | ALTER | AND | ANY | ARE | ARRAY | AS | ASC | ASSERTION | ASENSITIVE | AT | AUTHORIZATION | BEFORE | BEGIN | BIGINT | BIGSERIAL | BINARY | BIT | BLOB | BOOLEAN | BOTH | BREADTH | BREAK | BY | CALL | CASCADE | CASCADED | CASE | CAST | CATALOG |CHAR | CHECK | CLASS | CLOB | CLOSE | COLLATE | COLLATION | COLUMN | COMMIT | COMPLETION| CONDITION | CONNECT | CONT | CONNECTION | CONSTRAINT | CONSTRAINTS | CONSTRUCTOR | CONTINUE | CORRESPONDING | CREATE | CROSS | CUBE | CURRENT | CURRENT_DATE | CURRENT_PATH | CURRENT_ROLE | CURRENT_TIME | CURRENT_TIMESTAMP | CURRENT_USER | CURSOR | CYCLE| DATE | DAY | DEALLOCATE | DEC | DECIMAL | DECLARE | DEFAULT | DEFERRABLE | DEFERRED | DELETE | DEPTH | DEREF | DESC | DESCRIBE | DESCRIPTOR | DESTROY| DESTRUCTOR | DETERMINISTIC | DICTIONARY | DIAGNOSTICS | DISCONNECT | DISTINCT |DO |DOMAIN | DOUBLE | DROP | DYNAMIC | EACH | ELSE | ELSEIF | END | END-EXEC | EQUALS | ESCAPE | EVERY | EXCEPT| EXCEPTION | EXEC | EXECUTE | EXIT | EXTERNAL | FALSE | FETCH | FIRST | FLOAT | FOR | FOREIGN | FOUND | FROM| FREE | FULL | FUNCTION | GENERAL | GET | GLOBAL | GO | GOTO | GRANT | GROUP | GROUPING | HANDLER | HAVING | HOLD | HOST | IDENTITY | IF | IGNORE | IMMEDIATE | IN | INDICATOR | INITIALIZE | INITIALLY | INNER | INOUT | INPUT | INSENSITIVE | INT | INTEGER | INTERSECT | INTO | IS | ISOLATION | ITERATE | JOIN | KEY |

LANGUAGE | LANGUAGE SQL | LARGE | LAST | LATERAL | LEADING | LEAVE | LESS | LEVEL | LIKE | LIMIT | LOCAL | LOCALTIME | LOCALTIMESTAMP | LOCATOR | LOOP | MAP | MATCH | MODIFIES | MODIFY | MODULE | NAMES | NATIONAL | NATURAL | NCHAR | NCLOB | NEXT | NO | NONE | NOT | NULL | NUMERIC | NVARCHAR | OBJECT | OF | OFF | ON | ONLY | OPEN | OPERATION | OPTION | OR | ORDINALITY | OUT | OUTER | OUTPUT | PAD | PARTIAL | PATH | POSTFIX | PREFIX | PREORDER | PREPARE | PRESERVE | PRIMARY | PRIOR | PRIVILEGES | PROCEDURE | READ | READS | REAL | RECURSIVE | REFERENCES | REFERENCING | RELATIVE | REPEAT | RESTRICT | RESULT | RETURN | RETURNS | REVOKE | ROLE | ROLLBACK | ROLLUP | ROUTINE | ROW | ROWS|SAVEPOINT | SCHEMA | SCROLL | SCOPE | SEARCH | SECTION | SELECT| SENSITIVE | SEQUENCE | SERIAL | SESSION | SESSION_USER | SET | SETS | SHORT | SIZE | SMALLINT | SOME | SPECIFIC | SPECIFICTYPE | SQL | SQLCODE | SQLEXCEPTION | SQLSTATE | SQLWARNING | START | STATIC | STATISTICS | STOP | STRUCTURE | SYSTEM_USER | TABLE | TEMPORARY | TERMINATE | THAN | THEN | TIME | TIMESTAMP | TIMEZONE_HOUR| TIMEZONE_MINUTE | TO | TRACE | TRAILING | TRANSACTION | TRANSLATION | TREAT | TRIGGER | TRUE | UNDER | UNION | UNKNOWN | UNTIL | UNNEST | UPDATE | USAGE | USING | VALUE | VALUES | VARBINARY | VARBPTR | VARCHAR | VARCPTR | VARIABLE | VARYING | VIEW | WHEN | WHENEVER | WHERE | WHILE | WITH | WITHOUT | WORK | WRITE | ZONE

3 SQL文

DBMasterは包括的なSQL問合せ言語を提供します。SQL（Structured Query Language）は、ANSIによって標準化された問合せ言語です。現在の標準はANSI-99 SQLです。この章は、DBMasterがサポートする全てのANSI-99 SQL文を説明します。

3.1 ABORT BACKUP

目的

オンライン・バックアップの操作を中止します。

構文

説明

ABORT BACKUP文は、オンライン・バックアップをキャンセルします。バックアップ中にエラーが発生したり、バックアップを別の時点に取るときには、実行中のバックアップをキャンセルします。SYSADMまたはDBAセキュリティ権限をもつユーザーだけがABORT BACKUPを実行することができます。

バックアップ・モードは、オンライン増分バックアップを取るかどうか、どのデータをバックアップするかを指示します。3つのバックアップ・モード、NONBACKUP、BACKUP-DATA、BACKUP-DATA-AND-BLOBがあります。バックアップ・モードは、**dmconfig.ini**ファイルのDB_BMODEキーワード、dmSQLコマンド・プロンプトのSQLのSET文、JConfiguration Tool、JServer Managerのいずれかを使用して設定します。

NONBACKUPモードは、前回の完全バックアップ以降に挿入/更新されたデータを保護しません。データベースのプログラム障害はジャーナルを使用すれば完全にリカバリすることができますが、ディスク障害はデータの紛失につながるかもしれません。アクティブ・トランザクションが使用していないジャーナル・ブロックは、チェックポイントを取った後、直ちに再利用されます。一度上書きされると、データベースは前回の完全バックアップの時点にリストアできるだけになります。

BACKUP-DATAモードは、前回の完全バックアップ以降に挿入/更新されたBLOB以外のデータを保護します。オンライン増分バックアップを取り、BLOB以外のデータをバックアップ・ファイルに保存することができます。データベースのプログラム障害はジャーナルを使用して完全にリカバリされ、ディスク障害はBLOB以外のデータだけがリカバリされます。アクティブ・トランザクションが使用していないジャーナル・ブロックは、チェックポイントを取り、且つジャーナル・ファイルがバックアップされた後に再利用されます。

BACKUP-DATA-AND-BLOBモードは、前回の完全バックアップ以降に挿入/更新されたBLOBを含む全てのデータを保護します。オンライン増分バックアップを取り、全てのデータをバックアップ・ジャーナル・ファイルに保存することができます。データベースのプログラム障害はジャーナルを使用すれば完全にリカバリすることができ、ディスク障害も完全にリカバリすることができます。前回のバックアップを使用して、BLOBデータを含めて、メディア障害時点のデータベースに完全にリストアします。アクティブ・トランザクションが使用していないジャーナル・ブロックは、チェックポイントを取り、且つジャーナル・ファイルがバックアップされた後に再利用されます。

ABORT BACKUPを実行しても、データベースのバックアップ・モードは変わりません。データベースは、バックアップの開始前と同じバックアップ・モードのままです。

使用例

➲ **使用例**　バックアップ操作を中止します：

```
dmSQL> BEGIN BACKUP;
dmSQL> ABORT BACKUP;
```

関連SQL

BEGIN BACKUP

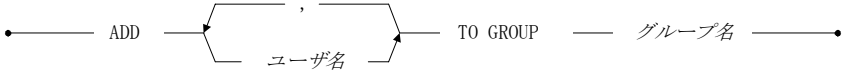
END BACKUP

3.2 ADD TO GROUP

目的

グループにユーザーを追加します。

構文



ユーザー名　　CONNECT以上の権限をもつ既存のユーザーの名前

グループ名　　既存のグループの名前

説明

ADD TO GROUP文は、既存のグループにユーザーを追加します。追加されたユーザーは、グループがもつ現在以降の全てのオブジェクト権限を獲得します。SYSADMまたはDBA権限をもつユーザーだけがこのSQL文を実行することができます。

多数のユーザーがいるデータベースの場合、グループを利用すると、オブジェクト権限の管理が簡単になります。グループに与えられたオブジェクト権限は、自動的にグループの全ユーザーに与えられます。

グループに追加されたユーザーは、元々もっていた権限も保持します。グループから削除されたユーザーは、グループのオブジェクト権限を失いますが、その他の直接あるいは他のグループによって与えられた権限はもち続けます。

ユーザー名の代わりにグループ名を指定することができます。ただし、グループ名の参照が循環してはいけません。ユーザー名とグループ名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1** ユーザー**Joe**と**John**をグループ**Manager**に追加します：

```
ADD Joe, John TO GROUP Manager;
```

➲ **使用例2** **FullTime**と**PartTime**グループをグループ**Staff**に追加します：

```
ADD FullTime, PartTime TO GROUP Staff;
```

➲ **使用例3** ユーザー**Bill**、グループ**FlexTime**をグループ**Staff**に追加します：

```
ADD Bill, FlexTime TO GROUP Staff;
```

関連SQL

CREATE GROUP

REMOVE FROM GROUP

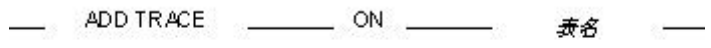
DROP GROUP

3.3 ADD TRACE

目的

単一表にトレースを追加して詳細な既存/新規データのログを取得します。

構文



表名　実在する単一の表の名前

説明

実際には3つの内部トリガーによって動作し、トレース対象の表のinsert/update/deleteオペレーションが記録されます。既存/新規データはDBNAME_currentdate_###.TXTに追加情報として記録されます。DBAまたはSYSADMである表の所有者のみがADD TRACEコマンドを使用できます。

注: DB_LGSVRの値は4以上でなければなりません。定義が4以上でなければ詳細情報は省略されログファイルには何も書き出されません。

使用例

➲ **使用例：**

表**tb1**にトレースを追加して、レコードを挿入、更新、削除します。

```
dmSQL> add trace on tb1;
dmSQL> insert into tb1 values (1, ‘abc’);
1 rows inserted
dmSQL> update tb1 set c2 = ‘xyz' where c1=1;
```

```
1 rows updated
dmSQL> delete from tb1;
1 rows deleted
```

3.4 ALTER DATAFILE

目的

データベースを構成するファイルのサイズを拡大します。

構文

ファイル名　　拡大するファイルの論理ファイル名

ページ数　　ファイルに追加するページ数

説明

ALTER DATAFILE文は、データファイルまたはBLOBファイルに指定したページ数を追加して拡大します。SYSADMまたはDBA権限をもつユーザーだけがALTER DATAFILEを実行することができます。

ファイルは、データベース内のデータを格納する物理ストレージです。ファイルはオペレーティング・システムが管理し、ファイル内のデータはDBMSが管理します。DBMasterは、データ、BLOB、ジャーナルの3種類のファイルを使用します。

データファイルとBLOBファイルは、ユーザーとシステムのデータを格納します。両ファイルとも類似の特性をもっていますが、パフォーマンスを上げるためにDBMasterはデータとBLOBを別々の方法で管理します。データファイルには表と索引のデータを格納し、一方、BLOBファイルにはバイナリラージオブジェクト（BLOB）を格納します。

ジャーナル・ファイルは、データベースの全ての変更履歴と変更ステータスをリアルタイムに記録する特殊なファイルです。ジャーナルは、失敗したトランザクションによる変更のundoや、成功したけれどもデータベース損傷によってディスクに書き込まれていない変更のredoを可能にします。ジャーナル・ファイルは、データベース管理システムによってのみ使用されます。ユーザーデータを格納するのに使用されることはありません。

データベースのデータ独立性を確保するために、オペレーティング・システムのファイルを直接参照することはできなくしています。データベースの各ファイルは、物理ファイル名と論理ファイル名の2つの名前をもっています。物理ファイル名はオペレーティング・システムが使用する名前であり、論理ファイル名はデータベースが使用する名前です。物理ファイル名と論理ファイル名は、**dmconfig.ini**ファイル内のエントリを経由して相互に作用します。

ALTER DATAFILEを使用するときは、論理ファイル名を指定します。ファイルの合計ページ数が2,147,483,647以下でディスク容量が許す範囲内ならば、1～2,147,483,645のページをファイルに追加することができます。表領域内の全ファイルの合計サイズは8 TB以下です。

使用例

➲ **使用例1　dmconfig.ini**ファイル内の2つのデータベース・ファイルの入力例を示します。左側が論理ファイル名、右側が物理ファイル名です：

```
customer_data = d:\DBMaster\tutorial\database\custdata.db 500
customer_blob = d:\DBMaster\tutorial\database\custblob.bb 1000
```

➲ **使用例2　customer_data**ファイルに**1000**ページを追加します：

```
ALTER DATAFILE customer_data ADD 1000 PAGES;
```

➲ **使用例3　dmconfig.ini**ファイル内の**customer_data**ファイルのページ数が**1000**ページ増分されました：

```
customer_data = d:\DBMaster\tutorial\database\custdata.db 1500
customer_blob = d:\DBMaster\tutorial\database\custblob.bb 1000
```

関連SQL

CREATE TABLESPACE

ALTER TABLESPACE

3.5 ALTER INDEX RENAME

目的

インデックス名の変更。

構文

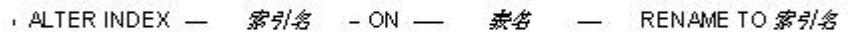


図3-4 ALTER Index Rename文字列

索引名——作成する新しいインデックスの名前

表名——インデックスを作成するテーブルの名前

説明

ALTER INDEX RENAMEコマンドは既存のテーブル上の既存のインデックスの名前を変更します。名前の変更はシステムカタログのインデックス名だけに影響し、データベースのインデックスは再構築しません。テーブルにコマンドを実行できるのは、テーブル所有者か、DBAか、INDEX権限を有するユーザーだけです。

使用例

➲ 使用例：

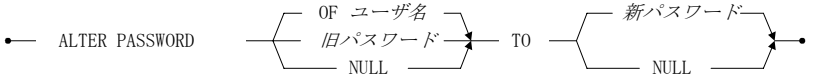
```
lter index ix1 on tb_tmp rename to ix_new;
```

3.6 ALTER PASSWORD

目的

ユーザーのパスワードを変更します。

構文



ユーザー名	パスワードを変更するユーザーの名前
旧パスワード	ユーザーの現在のパスワード
新パスワード	ユーザーの新しいパスワード

説明

ALTER PASSWORD文は、ユーザーの現在のパスワードを別のパスワードに変更します。ユーザーは自分自身のパスワードを変更することができ、SYSADM は全てのユーザーのパスワードを変更することができます。

自分のパスワードを変更するときは、ALTER PASSWORD *旧パスワード* TO *新パスワード* 文を使用します。SYSADMがユーザーのパスワードを変更するときは、ALTER PASSWORD OF *ユーザー名* TO *新パスワード* 文を使用します。SYSADMだけがこの形式のSQL文を使用することができます。

旧パスワードは、データベースに記憶されているユーザーのパスワードと一致しなければなりません。現在ユーザーがパスワードをもっていないときは、旧パスワードをNULLにします。パスワードを削除するときは、新パスワードをNULLにします。

パスワードの最大長は16字です。文字、数字、アンダースコア、記号$と#を使用することができ、最初は数字以外にします。

使用例

- **使用例1**　パスワードの無いユーザーが自分のパスワードを**abcdef**にする：

```
ALTER PASSWORD NULL TO abcdef;
```

- **使用例2**　パスワードを**abcdef**から**a23456**に変更します：

```
ALTER PASSWORD abcdef TO a23456;
```

- **使用例3**　パスワード**a23456**を削除します：

```
ALTER PASSWORD a23456 TO NULL;
```

- **使用例4**　**SYSADM**はユーザー**John**のパスワードが何であっても無条件に**abcedf**に変更することができます：

```
ALTER PASSWORD OF John TO abcdef;
```

関連SQL

CONNECT

GRANT (セキュリティ権限)

REVOKE (セキュリティ権限)

CREATE GROUP

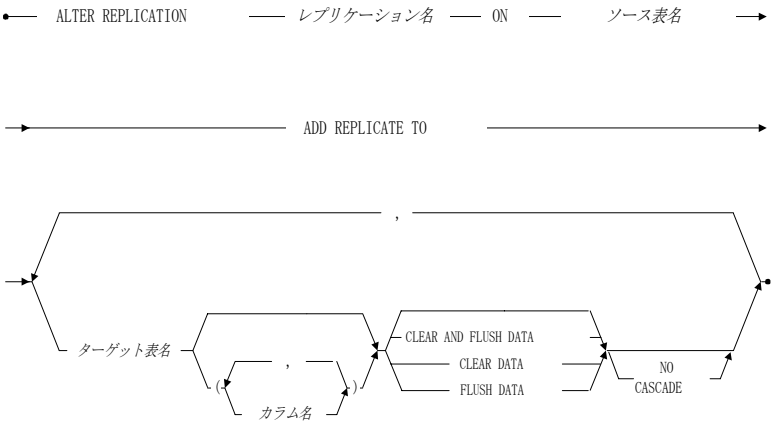
REMOVE FROM GROUP

3.7 ALTER REPLICATION ADD REPLICATE

目的

既存の表レプリケーションにターゲット表を追加します。

構文



レプリケーション名	ターゲット表を追加するレプリケーションの名前
ソース表名	レプリケーションが作成されたソース表の名前
ターゲット表名	ターゲット・データベースにある表の名前
カラム名	レプリケートするターゲット表のカラムの名前

説明

ALTER REPLICATION ADD REPLICATE文は、既存の表レプリケーションにターゲット表を追加します。複数のターゲット表を追加することができます。ALTER REPLICATION ADD REPLICATEは、ソース表所有者、DBA、SYSADMが実行することができます。

表レプリケーションは、表全体または一部のカラムを遠隔地にコピーします。遠隔地のユーザーは、コピーされたデータで作業できるようになります。レプリケートされたコピーは、他の場所のデータベースと同期が取られます。各データベースは、低速ネットワーク接続を経由して別のマシンに行かずに、直接効率的にデータ要求に応えることができるようになりま

す。レプリケーションは、ターゲット側にデータベースのバックアップを取るのとは異なり、ユーザーの介入無しにDBMS自身によってトランザクション毎に同期が取られます。

レプリケーションには同期と非同期があります。同期レプリケーションは、ソース表の変更と同時にターゲット表を変更します。非同期レプリケーションは、ソース表の変更を格納しておき、予め定義されたスケジュールに従ってターゲット表を変更します。ALTER REPLICATION ADD REPLICATEは、同期および非同期のレプリケーションを変更してターゲット表を追加します。

DBMasterの同期レプリケーションは、グローバルトランザクションモデルを使用します。このモデルでは、ターゲット表へのデータ・レプリケーションがソース・トランザクションにとって不可欠な部分として取り扱われます。これは、ターゲット・データベースへのデータ・レプリケーションに失敗すると、ソース表のトランザクションも失敗することを意味します。

トランザクションは、データベースの整合性を維持するための論理的な作業単位であり、伝統的に、一括して完了しなければならない一連のデータベースオペレーションと定義されています。トランザクションは一つの統合体であり、全てのデータを変更して完了するか、全てのデータを変更しないで失敗するか、のどちらかでなければなりません。

DBMasterの非同期レプリケーションは、トランザクションログを使用してターゲット表にデータをコピーします。ソース表の変更はトランザクションログに格納され、予め決められたスケジュールに従ってターゲット表にコピーされます。ソース・トランザクションとターゲット・トランザクションは、トランザクションログを使用することによって独立に取り扱われ、ターゲットに接続できなくてもソース表を更新できるようになります。非同期レプリケーションは、ネットワークやターゲット・データベースの障害にも耐えることができます。レプリケーションは、障害が復旧するまで再試行し続けます。

レプリケーションを変更しターゲット表を追加するには、レプリケーション名、ソース表名、ターゲット表名を指定します。ソース表とターゲット表は、各々のデータベースに存在するものでなければなりません。表が削

除されると、その表に作成されているレプリケーションも自動的に削除されます。

レプリケーション作成時にソース表のカラムリストを指定しないと表全体がレプリケートされます。ソース表のカラムリストは、レプリケーション作成時にだけ指定することができます。カラムリストを与えずに表全体をレプリケートする場合、ソースおよびターゲットの表のカラムは、同じ名前とデータ型をもっていなければなりません。

ソースとターゲットの表のカラム名が異なるときは、ターゲット表のカラムリストを指定します。ソース表のカラムは、左から右に対応するターゲット表のカラムリストのカラムにレプリケートされます。ソースとターゲットの両方のカラムリストを与えて、ソース表とターゲット表の対応カラムを明示的に指定します。両方の表の主キーカラムは、カラム数とデータ型が一致していなければなりません。

DBMasterは、所有者名とオブジェクト名を組み合わせた完全修飾名でレプリケーションを識別することはせず、代わりにレプリケーションを表に関連付けます。このために、同じ表上の全てのレプリケーション名は一意でなければなりません。

同期レプリケーションは、ソース表の所有者のセキュリティ権限とオブジェクト権限で動作します。ターゲット表がリンクで指定されている場合は、レプリケーションはリンクと同じセキュリティ権限とオブジェクト権限で動作します。

非同期レプリケーションは、CREATE SCHEDULE文のIDENTIFIED BY句で指定したリモートアカウントのセキュリティ権限とオブジェクト権限で動作します。非同期レプリケーションのスケジュールは、レプリケーションを作成する前に作成しておきます。

CLEAR DATA／FLUSH DATA／CLEAR AND FLUSH DATAキーワードはオプションです。これらのキーワードは、レプリケーション作成時に実行するオペレーションを指定します。CLEAR DATAキーワードは、ターゲット表の全データを削除します。FLUSH DATAキーワードは、検索条件にマッチするデータをターゲット表にコピーします。CLEAR AND FLUSH DATAキーワードは、ターゲット表の全データを削除してから検索条件に

マッチするデータをターゲット表にコピーします。オプションを指定しないと、何も行われません。

NO CASCADEキーワードはオプションです。このキーワードは、カスケードレプリケーションかどうかを指定します。通常、指揮系統が上位から下位に流れるように、データをAからBに、次にCにレプリケートするのが、典型的なカスケードレプリケーションです。非カスケードモデルでは、AはBにデータをレプリケートして、それ以上は別のデータベースにデータをレプリケートしません。このようなデータモデルには、NO CASCADEオプションを指定します。初期設定の指定はCASCADEです。

使用例

- **使用例1**　ソース表**Employeesinfo**に作成された同期または非同期レプリケーション**EmpRep**を変更します。ソース**dmconfig.ini**ファイルのデータベース構成セクション名**Div1Office**で識別されるターゲット・データベースの表**Div1Emp**へのデータ・レプリケーションを追加します。表**Employees**と**Div1Emp**の全てのカラム名とデータ型は同じです：

```
ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div1Office:Div1Emp;
```

- **使用例2**　使用例1と同じですが、**CLEAR DATA**キーワードにより、レプリケーションを作成する前にターゲット表の全データが削除されます：

```
ALTER REPLICATION EmpRep ON Employees info ADD REPLICATE TO
                            Div1Office:Div1Emp CLEAR DATA;
```

- **使用例3**　使用例1と同じですが、**FLUSH DATA**キーワードにより、レプリケーションを作成する前にソース表にある全データがターゲット表に送られます：

```
ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div1Office:Div1Emp FLUSH DATA;
```

➲ 使用例4　使用例**1**と同じですが、**CLEAR AND FLUSH DATA**キーワードにより、レプリケーションを作成する前にターゲット表の全データが削除され、次にソース表の全データがターゲット表に送られます：

```
ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div1Office:Div1Emp CLEAR AND FLUSH DATA;
```

➲ 使用例5　上の使用例と類似していますが、ターゲット・データベース**Div2Office**の**Div2Emp**表と、ターゲット・データベース**Div3Office**の**Div3Emp**表へのレプリケーションを追加します。ソース**dmconfig.ini**ファイルには、両方のターゲット・データベースと同名のデータベース構成セクションがあります：

```
ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div2Office:Div2Emp CLEAR DATA,
                            Div3Office:Div3Emp FLUSH DATA;
```

関連SQL

ALTER REPLICATION DROP REPLICATE

ALTER SCHEDULE

CREATE REPLICATION

CREATE SCHEDULE

DROP REPLICATION

DROP SCHEDULE

SUSPEND SCHEDULE

RESUME SCHEDULE

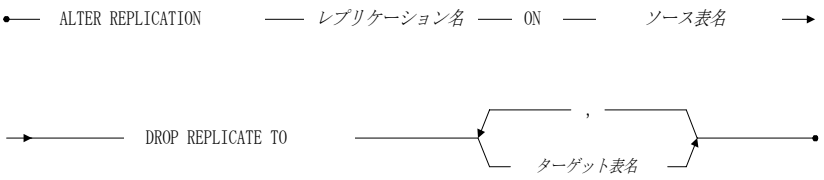
SYNCHRONIZE SCHEDULE

3.8 ALTER REPLICATION DROP REPLICATE

目的

既存のレプリケーションからターゲット表を削除します。

構文



レプリケーション名	ターゲット表を削除するレプリケーションの名前
ソース表名	レプリケーションが作成されているソース表の名前
ターゲット表名	レプリケートを中止するターゲット表の名前

説明

ALTER REPLICATION DROP REPLICATE文は、既存のレプリケーションからターゲット表を削除します。ターゲット表にデータをレプリケートしたくないときには、レプリケーションからその表を削除します。ALTER REPLICATION DROP REPLICATEは、ソース表の所有者、DBA、SYSADMだけが実行することができます。

レプリケーションは、表全体または一部のカラムを遠隔地にコピーします。遠隔地のユーザーは、コピーされたデータで作業できるようになります。レプリケートされたコピーは、他の場所のデータベースと同期が取られます。各データベースは、低速ネットワーク接続を経由して別のマシン

に行かずに、直接効率的にデータ要求をサービスできるようになります。レプリケーションは、ターゲット側にデータベースのバックアップを取るのとは異なり、ユーザーの介入無しにDBMS自身によってトランザクション毎に同期が取られます。

レプリケーションには同期と非同期があります。同期レプリケーションは、ソース表の変更と同時にターゲット表を変更します。非同期レプリケーションは、ソース表の変更を格納しておき、予め定義されたスケジュールに従ってターゲット表を変更します。ALTER REPLICATION DROP REPLICATEは、同期および非同期のレプリケーションを変更しターゲット表を削除します。

DBMasterの同期レプリケーションは、グローバルトランザクションモデルを使用します。このモデルでは、ターゲット表へのデータ・レプリケーションがソース・トランザクションの不可欠な部分として取り扱われます。トランザクションは、データベースの整合性を維持するための論理的な作業単位であり、伝統的に、一括して完了しなければならない一連のデータベースオペレーションと定義されています。トランザクションは一つの統合体であり、全てのデータを変更して完了するか、全てのデータを変更しないで失敗するか、のどちらかでなければなりません。これは、ターゲット・データベースへのデータ・レプリケーションに失敗すると、ソース表のトランザクションも失敗することを意味します。

DBMasterの非同期レプリケーションは、トランザクションログを使用してターゲット表にデータをレプリケートします。ソース表の変更はトランザクションログに格納され、予め決められたスケジュールに従ってターゲット表にレプリケートされます。ソース・トランザクションとターゲット・トランザクションは、トランザクションログを使用することによって独立に取り扱われ、ターゲットに接続できなくてもソース表を更新できるようになります。非同期レプリケーションは、ネットワークやターゲット・データベースの障害にも耐えることができます。レプリケーションは、障害が復旧するまで再試行されます。

レプリケーションからターゲット表を削除するには、レプリケーション名、ソース表名、ターゲット表名を指定します。複数のターゲット表をレプリケーションから削除するには、削除する全ての表をリストします。表

が削除されると、表に作成されているレプリケーションも自動的に削除されます。

使用例

➲ **使用例1**　ソース表**Employeesinfo**に作成されたレプリケーション**EmpRep**からターゲット表**Div1Emp**を削除します：

```
ALTER REPLICATION EmpRep ON Employeesinfo DROP REPLICATE TO Div1Emp;
```

➲ **使用例2**　ソース表**Employeesinfo**に作成されたレプリケーション**EmpRep**からターゲット表**Div2Emp**、**Div3Emp**、**Div4Emp**を削除します：

```
ALTER REPLICATION EmpRep ON Employeesinfo
                        DROP REPLICATE TO Div2Emp, Div3Emp, Div4Emp;
```

関連SQL

ALTER REPLICATION ADD REPLICATE

ALTER SCHEDULE

CREATE REPLICATION

CREATE SCHEDULE

DROP REPLICATION

DROP SCHEDULE

SUSPEND SCHEDULE

RESUME SCHEDULE

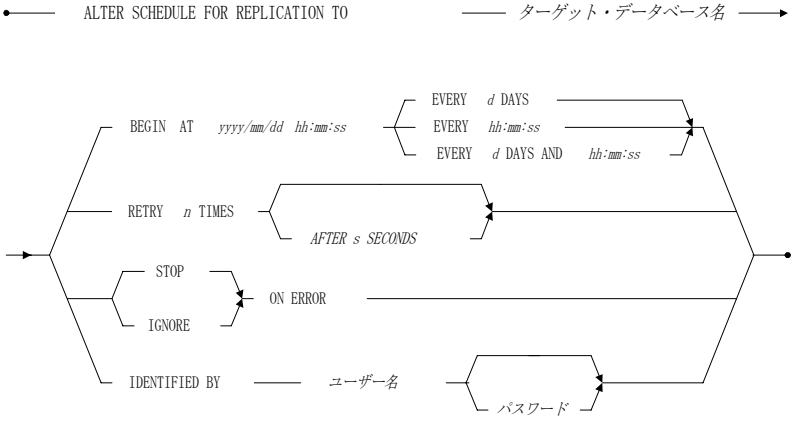
SYNCHRONIZE SCHEDULE

3.9 ALTER SCHEDULE

目的

非同期レプリケーションのスケジュールを変更します。

構文



ターゲット・データベース名	レプリケーション・スケジュールを作成するターゲット・データベースの名前 (リンク名は不可)
yyyy/mm/dd	レプリケーションの開始日
hh:mm:ss	1. レプリケーションの開始時刻 2. レプリケーションの間隔時間
d	レプリケーション間隔の日数
n	失敗したときの再試行回数
s	失敗したときの再試行間隔の秒数
ユーザー名	ターゲット・データベースのユーザー名
パスワード ワード	ターゲット・データベースユーザーのパスワード

説明

ALTER SCHEDULE文は、非同期レプリケーションのスケジュールを変更します。同期レプリケーションはスケジュールを使用しないので、ALTER SCHEDULEは同期レプリケーションには何の影響も与えません。DBAまたはSYSADMセキュリティ権限をもつユーザーだけがALTER SCHEDULEを実行することができます。

レプリケーションは、表全体または一部のカラムを遠隔地にコピーします。遠隔地のユーザーは、コピーされたデータで作業できるようになります。レプリケートされたコピーは、他の場所のデータベースと同期が取られます。各データベースは、低速ネットワーク接続を経由して別のマシンに行かずに、直接効率的にデータ要求に応じることができるようになります。レプリケーションは、ターゲット側にデータベースのバックアップを取るのとは異なり、ユーザーの介入無しにDBMS自身によってトランザクション毎に同期が取られます。

レプリケーションには同期と非同期があります。同期レプリケーションは、ソース表の変更と同時にターゲット表を変更します。非同期レプリケーションは、ソース表の変更を格納しておき、予め定義されたスケジュールに従ってターゲット表を変更します。ALTER SCHEDULEは、非同期レプリケーションにのみ有効です。

BEGIN ATは、非同期レプリケーションを開始する日時を指定します。日付はyyyy/mm/ddフォーマットで指定し、yyyyは1970～2038の年、mmは01～12の月、ddは01～31の日です。時刻は*hh:mm:ss*フォーマットで指定し、*hh*は 00～23の時、*mm*は00～59の分、*ss*は00～59の秒です。年は1970～2038の範囲になければなりません。BEGIN ATキーワードには日付と時刻を含めます。レプリケーションが既に作動した後にレプリケーション開始日時を将来の日時に変更すると、ターゲット・データベースにレプリケートされていない表データは、新しいレプリケーション開始日時まで待たされます。

EVERYは、非同期レプリケーションの間隔を定義します。レプリケーション間隔は、時間:分:秒、日数、または両方の組み合わせで与えることができます。EVERY *hh:mm:ss*の*hh*は00～23の時間数、*mm*は00～59の分数、*ss*

は00～59の秒数です。EVERY *d* DAYSは日数を指定します。*d*は1～365の日数です。EVERY *d* DAYS AND *hh:mm:ss* は、日数と時間数を組み合わせて指定します。

RETRYは、SQL文実行中のエラー、例えばロック・タイムアウト、ジャーナルフルに起因するセーブポイントへのロールバック等が発生したときに、表データ・レプリケーションの再試行回数を指示します。RETRY *n* TIMESの*n*は0～2,147,483,647の数です。初期設定値は0です。

ネットワークエラーやターゲット・データベースエラーによってターゲット・サーバーに接続できないときは、次のレプリケーション・スケジュールの日時まで待ってレプリケートされていない表データを送信します。ロールバックが必要なトランザクションエラーは一度だけ再試行します。これに失敗すると、次のレプリケーション・スケジュールの日時まで待ちます。

AFTERキーワードはオプションです。このキーワードはRETRYキーワードと一緒に使用し、エラー発生時の再試行間隔を指定します。AFTER *s* SECONDSの*s*は0～2147483647の秒数です。初期設定値は5です。

ON ERRORキーワードは、ターゲット・データベースのデータがレプリケートできないように更新されているときのアクションを指定します。例えば、既にターゲット表から削除されたレコードを削除しようとしたり、既に存在するレコードをターゲット表に挿入しようとした場合です。この種のエラーが発生したときに、STOP ON ERRORとIGNORE ON ERRORの二つのオプションがあります。STOP ON ERRORは、データのレプリケーションを止める指示です。IGNORE ON ERRORは、エラーデータを無視し残りのデータのレプリケーションを続ける指示です。初期設定の動作はIGNORE です。

IDENTIFIED BYキーワードは、ターゲット・データベースに接続するときに使用するユーザー名とパスワードを指定します。指定されたターゲット・データベースのユーザーは、ターゲット表にINSERT、DELETE、UPDATEする十分な権限をもたなければなりません。ユーザーに与えられているセキュリティ権限とオブジェクト権限が実行できるオペレーションを決定します。

スケジュールを変更するターゲット・データベース名を指定します。データベースリンク名を指定してはいけません。ターゲット・データベース上の全ての非同期レプリケーションは、変更されたスケジュールを使用します。

使用例

➲ **使用例1**　ターゲット・データベース**EmpRep**の非同期レプリケーション・スケジュールを変更します。ロック・タイムアウトまたはジャーナルフルに起因するセーブポイントへのロールバックの再試行回数を3回にし、再試行間隔を5秒にします：

```
ALTER SCHEDULE FOR REPLICATION TO EmpRep
                RETRY 3 TIMES AFTER 5 SECONDS;
```

➲ **使用例2**　ターゲット・データベース**EmpRep**の非同期レプリケーションスケジュールを変更します。ターゲット・データベースのデータがレプリケートできないように更新されているときのアクションを**STOP**にします：

```
ALTER SCHEDULE FOR REPLICATION TO EmpRep STOP ON ERROR;
```

➲ **使用例3**　ターゲット・データベース**EmpRep**の非同期レプリケーション・スケジュールを変更します。ターゲット・データベースに接続するときのユーザー名とパスワードを新しいものに設定します：

```
ALTER SCHEDULE FOR REPLICATION TO EmpRep IDENTIFIED BY RepUser rdejpe88;
```

関連SQL

ALTER REPLICATION ADD REPLICATE

ALTER REPLICATION DROP REPLICATE

CREATE REPLICATION

CREATE SCHEDULE

DROP REPLICATION

DROP SCHEDULE

SUSPEND SCHEDULE

RESUME SCHEDULE

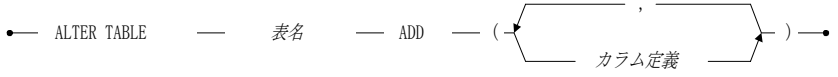
SYNCHRONIZE SCHEDULE

3.10 ALTER TABLE ADD COLUMN

目的

表に新規カラムを追加します。

構文



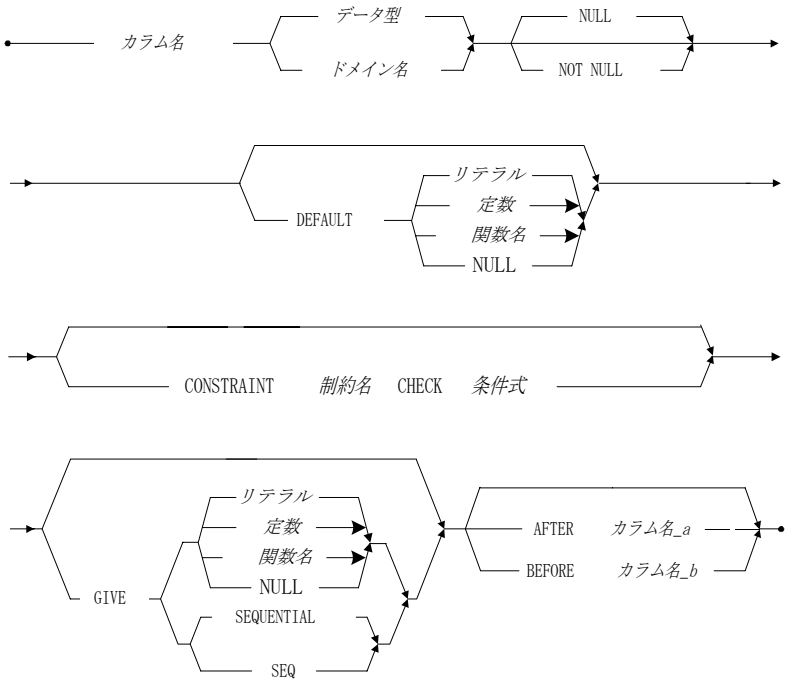
表名	カラムを追加する表の名前
カラム定義	追加するカラムの定義

説明

ALTER TABLE ADD COLUMN文は、既存の表定義に新規カラムを追加します。表所有者、DBA、表のALTER権限をもつユーザーだけがALTER TABLEを実行することができます。

カラム定義は、カラム名とデータ型またはドメインで指定します。複数のカラムを追加することができます。表の最大カラム数は2000です。カラム定義の構文とキーワードの使用法を以下に示します。

カラム定義



カラム名	新規カラムの名前
データ型	新規カラムのデータ型
ドメイン名	新規カラムのドメインの名前
リテラル	値が挿入されないときに使用するリテラル値
定数	値が挿入されないときに使用する定数
関数名	値が挿入されないときに使用する組み込み関数
制約名	カラムに設ける制約の名前
条件式	評価結果が真または偽となる式
カラム名_a	新規カラムの直前にある既存カラムの名前

カラム名_b　　新規カラムの直後にある既存カラムの名前

DBMasterは次のデータ型をサポートします。BIGINT、BIGSERIAL、BINARY、CHAR、DATE、DECIMAL（NUMERIC）、DOUBLE、FILE、FLOAT、INTEGER、LONG VARBINARY（BLOB）、LONG VARCHAR（CLOB）、OID、SERIAL、SMALLINT、TIME、TIMESTAMP、VARCHAR。

データ型の代わりにユーザー定義ドメインをカラムに指定することができます。ドメインは、データ型、初期設定値、制約の組み合わせです。ドメインを使用してカラムを定義すると、これらがカラムに適用されます。初期設定値と制約については、以下のDEFAULTとCHECKキーワードの説明を参照してください。カラム定義で初期設定値と制約を指定すると、ドメインの初期設定値と制約は打ち消されます。カラム定義でドメインの制約に制約を追加することもできます。

NULL／NOT NULLキーワードはオプションです。これらのキーワードは、新しい行を挿入するときにカラムを空白のままにできるかどうかを指定します。NULLキーワードは、行を挿入するときにカラム値を未定にしてもよい指定です。NOT NULLキーワードは、行を挿入するときにカラム値を挿入しなければならない指定です。NOT NULLは、表に何も無いときだけ指定することができます。既存の行は新規カラムの値をもたないのでNOT NULLの規則に違反するからです。誤ってNOT NULLを指定すると、カラムの追加は失敗します。

DEFAULTキーワードはオプションです。このキーワードは、新しく挿入する行のカラム値が空のときにカラムに挿入される初期設定値を指定します。初期設定値としては、リテラル値、定数、組み込み関数値、NULLキーワードを指定することができます。組み込み関数は、PI( )、NOW( )、USER( )のように引数の無い関数を使用します。DEFAULT値がNULLのときは、カラムをNOT NULLに定義することはできません。データ型の代わりにユーザー定義のドメインを使用するときは、通常、DEFAULTキーワードは使用しません。DEFAULT句はドメインに含まれるのが普通です。

CHECKキーワードはオプションです。このキーワードは、カラムに入力できる値の範囲を指定します。真または偽と評価される任意の条件式で許容値の範囲を指定することができます。CHECK条件式の中では、カラムの値

を表すのにVALUEキーワードを使用することができます。SQL文がCHECK条件を満たさないときは、そのSQL文は処理されません。データ型の代わりにドメインを使用するときは、通常、CHECKキーワードは使用しません。CHECK句はドメインに含むのが普通です。

GIVEキーワードはオプションです。このキーワードは、表の既存の行に挿入する新規カラムの値を指定します。GIVEキーワードを指定しないと既存の行の新規カラムはNULLになります。SERIALデータ型のカラムはNULL値をもつことができないのでSERIALカラムを追加するときはGIVEキーワードを指定します。GIVE値としては、リテラル値、定数、組み込み関数値、NULLキーワードを指定することができます。GIVE値としてNULLを使用するときは、カラムをNOT NULLで定義することはできません。SERIALカラムを追加するときは、GIVEキーワードだけでなくSEQUENTIAL／SEQキーワードも指定します。SEQUENTIAL／SEQは、SERIALデータ型のカラム定義で指定した値から始まるシリアル番号を既存の行に挿入する指定です。シリアル番号は新規行の挿入毎に増分されます。

BEFORE／AFTERキーワードはオプションです。これらのキーワードは、新規カラムの挿入位置と既存カラムとの関係を指定します。BEFOREキーワードは、指定した既存カラムの前（左）に新規カラムを挿入します。AFTERキーワードは、指定した既存カラムの後（右）に新規カラムを挿入します。BEFORE／AFTERキーワードを指定しないと、新規カラムは表の右端に追加されます。

新規カラムを表に追加しても、表に定義されているビューやシノニムは影響を受けません。カラム名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1**　表**Employeesinfo**に**DATE**データ型のカラム**HireDate**を追加します：

```
ALTER TABLE Employeesinfo ADD (HireDate DATE);
```

➲ **使用例2**　使用例1の**HireDate**カラムに**NOT NULL**キーワードを指定して、新規行を挿入するときには必ず**HireDate**の値を入力させます：

```
ALTER TABLE Employeesinfo ADD (HireDate DATE NOT NULL);
```

➲ **使用例3**　使用例2の**HireDate**カラムに**DEFAULT**キーワードを指定して、値を入力しないときに挿入する初期設定値を指定します。この場合だけ**NOT NULL**キーワードをもつカラム値の入力を省略することができます。この例では、組み込み関数**NOW( )**を使用してカラム値を指定しないときには現在の日付を挿入します：

```
ALTER TABLE Employeesinfo ADD (HireDate DATE NOT NULL DEFAULT NOW());
```

➲ **使用例4**　使用例3の**HireDate**カラムに**CHECK**キーワードを使用して入力できる値の範囲を指定します。**VALUE**キーワードは、カラムに入力された値を表します：

```
ALTER TABLE Employeesinfo ADD (HireDate DATE NOT NULL DEFAULT NOW() CHECK
VALUE > '01/01/1995');
```

➲ **使用例5**　**DATE** データ型の代わりにドメイン**D_ValidDates**を使用して同じ**HireDate**カラムを追加します。**DEFAULT**と**CHECK**句は、通常、ドメイン定義の中に含まれます。ドメインを使用するときは、**DEFAULT**と**CHECK**キーワードを使用しないのが普通です：

```
ALTER TABLE Employeesinfo ADD (HireDate D_ValidDates NOT NULL);
```

関連SQL

CREATE TABLE

ALTER TABLE DROP COLUMN

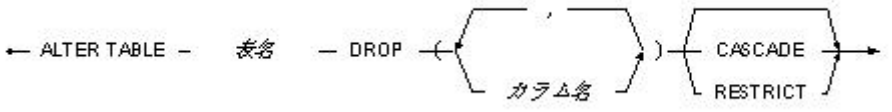
ALTER TABLE MODIFY COLUMN

3.11　ALTER TABLE DROP COLUMN

目的

表からカラムを削除します。

構文



表名	カラムを削除する表の名前
カラム名	削除するカラムの名前

説明

ALTER TABLE DROP COLUMN文は、既存の表に定義されているカラムを削除します。表の所有者、DBA、SYSADM、表のALTER権限をもつユーザーだけがALTER TABLE DROP COLUMNを実行することができます。

このSQL文は、不要になったカラムを表から削除するときに使用します。主キーまたは外部キーが定義されているカラムは、先に主キーまたは外部キーを削除しておかないと削除できません。ビューが定義されているカラムを削除すると、ビューは不正になり使用するとエラーが返されます。削除されたカラムのデータは、リカバリすることができません。十分に注意して使用してください。

CASCADE/RESTRICTのキーワードはオプションです。これらのキーワードは、削除されたカラムを参照する従属したオブジェクトを削除するか、確認するかを指定します。**CASCADE**のキーワードが指定されると、カラムに従属するオブジェクトはすべて削除されます。**RESTRICT**のキーワードが指定されると、ビュー定義や外部キーや制約で参照されるカラムは削除されません。**RESTRICT**のキーワードが指定されると、従属するオブジェクトがないカラムだけが削除されます。

使用例

➲ **使用例1　Employeesinfo**表の**BirthDate**カラムを削除します：

```
ALTER TABLE Employeesinfo DROP (BirthDate);
```

➲ **使用例2　Employeesinfo**表の**BirthDate**と**HireDate**カラムを削除します：

```
ALTER TABLE Employeesinfo DROP (BirthDate, HireDate);
```

➲ **使用例3**　以下のコマンドは**Employeesinfo**テーブルと従属するビュー**EmpView**からカラム**BirthDate**を削除します：

```
CREATE VIEW EmpView AS SELECT BirthDate FROM Employeesinfo.
ALTER TABLE Employeesinfo DROP (BirthDate) CASCADE;
```

関連SQL

ALTER TABLE ADD COLUMN

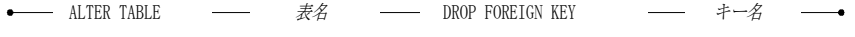
ALTER TABLE MODIFY COLUMN

3.12　ALTER TABLE DROP FOREIGN KEY

目的

表の外部キーを削除します。

構文



表名	外部キーを削除する表の名前
キー名	削除する外部キーの名前

説明

ALTER TABLE DROP FOREIGN KEY文は、既存の表に定義されている外部キーを削除します。表の所有者、DBA、表のALTER権限をもつユーザーだけがこのSQL文を実行することができます。

キーは、表内の特定の行の識別に役立てるカラム又はカラムの組み合わせです。キーを構成するカラムをキーカラムといいます。一意キーとは、重複するキー値をもつレコードが無いキーのことです。

主キーは、表内の行を一意的に識別するキーです。主キーが無いと行は重複値をもつ可能性があり、表内の特定の行を他と区別することが不可能になります。重複値をもつカラムを主キーに定義したり、既存の主キーに重複値を入力することはできません。

外部キーは、別の表の主キーまたは一意索引に対応するキーです。二つの表は、共通のキーデータによって親子関係が付けられます。親表は主キーまたは一意索引をもち、子表は外部キーをもちます。

参照整合性は、子表の外部キー (子キー) の値が親表の主キーまたは一意索引 (親キー) に対応する値をもつことを確実にします。参照整合性は、外部キーによって確立される親子関係を使用して表間に施行されます。DBMasterは、外部キーの定義を通して、自動的に表間の参照整合性制約をサポートします。子表にレコードを追加するときは、子キーの値が親キーに存在していなければなりません。同様に、親表からレコードを削除するときは、対応する子キーをもつ全てのレコードを先に削除しなければなりません。

参照アクションは、子キーが親キーを参照しているときに親キーを更新/削除するという、通常は参照整合性によって許されない手段を与えます。参照アクションは、親キーを更新/削除するときに対応する全ての子キーに実行するオペレーションを定義します。DBMasterは、更新/削除に対して4種の参照アクション、CASCADE、SET NULL、SET DEFAULT、NO ACTIONをサポートします。CASCADEは、対応する子キーを親キーと同様に更新/削除します。SET NULLは、対応する子キーをNULLに設定します。SET DEFAULTは、対応する子キーをカラムの初期設定値に設定します。NO ACTIONは、通常の参照整合性の規則に従います。外部キーを作成すると

きに参照アクションを定義しないと、NO ACTIONが初期設定のアクションになります。

外部キーが不要になったならば、ALTER TABLE DROP FOREIGN KEY文を使用して表から外部キーを削除します。外部キーを削除すると、もはや参照整合性は施行されず、子表に対する参照アクションも実行されません。外部キーが無いと、親表に無い値を子表に挿入したり、親表の値を更新/削除することが可能になります。このSQL文は十分に注意して使用すべきです。

使用例

➲ **使用例**　表**Salary**から外部キー**fkey**を削除します：

```
ALTER TABLE Salary DROP FOREIGN KEY fkey;
```

関連SQL

ALTER TABLE PRIMARY KEY

ALTER TABLE FOREIGN KEY

ALTER TABLE DROP PRIMARY KEY

3.13 ALTER TABLE DROP PRIMARY KEY

目的

表の主キーを削除します。

構文

表名　　主キーを削除する表の名前

説明

ALTER TABLE DROP PRIMARY KEY文は、表に定義されている主キーを削除します。表の所有者、DBA、表のALTERとINDEX権限をもつユーザーだけがこのSQL文を実行することができます。

キーは、表内の特定の行の識別に役立てるカラムまたはカラムの組み合わせです。キーを構成するカラムをキーカラムといいます。一意キーとは、重複するキー値をもつレコードが無いキーのことです。

主キーは、表内の行を一意的に識別するキーです。主キーが無いと、行は重複値をもつ可能性があるので表内の特定の行を他と区別することができなくなります。重複値をもつカラムを主キーに定義したり、既存の主キーに重複値を入力することはできません。

外部キーは、別の表の主キーまたは一意索引に対応するキーです。二つの表は、共通のキーデータによって親子関係が付けられます。親表は主キーまたは一意索引をもち、子表は外部キーをもちます。

参照整合性は、子表の外部キー (子キー) の値が親表の主キーまたは一意索引 (親キー) に対応する値をもつことを確実にします。参照整合性は、外部キーによって確立される親子関係を使用して表間に施行されます。DBMasterは、外部キーの定義を通して、自動的に表間の参照整合性制約をサポートします。子表にレコードを追加するときは、子キーの値が親キーに存在していなければなりません。同様に、親表からレコードを削除するときは、対応する子キーをもつ全てのレコードを子表から削除しておかなければなりません。

主キーが不要になったならば、ALTER TABLE DROP PRIMARY KEY文を使用して削除します。外部キーが定義されているときは、参照整合性が施行されています。主キーを削除する前に、主キーを参照する全ての外部キーを削除します。主キーを削除すると、各レコードの一意的なキー値は要求されなくなります。二つのレコードを区別することができないキー値を入力することができ、データベースの不整合を発生する可能性があります。このSQL文は十分に注意して使用すべきです。

使用例

➲ **使用例**　表**Employeesinfo**の主キー**PrimaryKey**を削除します：

```
ALTER TABLE Employeesinfo DROP PRIMARY KEY;
```

関連SQL

ALTER TABLE PRIMARY KEY

ALTER TABLE FOREIGN KEY

ALTER TABLE DROP FOREIGN KEY

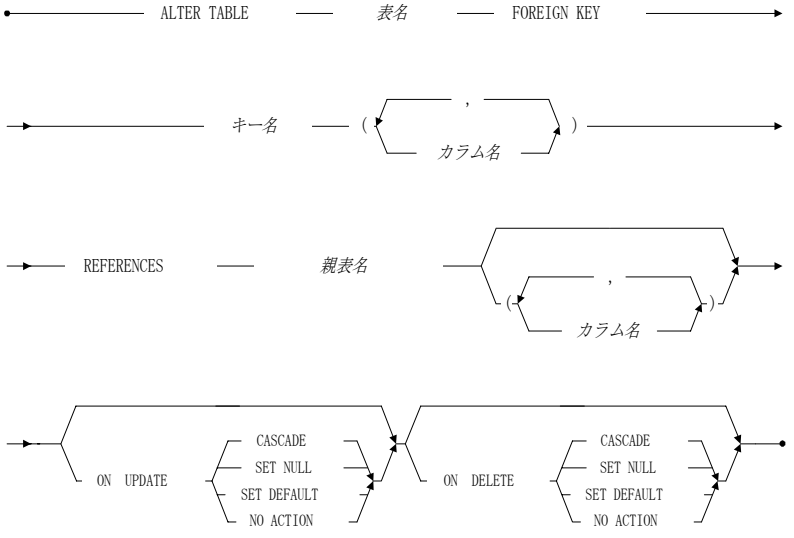
CREATE TABLE

3.14 ALTER TABLE FOREIGN KEY

目的

表に外部キーを追加します。

構文



表名	外部キーを追加する表の名前
キー名	追加する外部キーの名前
カラム名	1. 外部キーを作成するカラムの名前
	2. 外部キーが参照するカラムの名前
親表名	外部キーが参照する表の名前

説明

ALTER TABLE FOREIGN KEY文は、既存の表に外部キーの定義を追加します。ALTER TABLE FOREIGN KEYは、表の所有者、DBA、表のALTER権限と主キーの表またはカラムのREFERENCE権限をもつユーザーだけが実行することができます。

キーは、表内の特定の行の識別に役立てるカラムまたはカラムの組み合わせです。キーを構成するカラムをキーカラムといいます。一意キーとは、重複するキー値をもつレコードが無いキーのことです。

主キーは、表内の行を一意的に識別するキーです。主キーが無いと、行は重複値をもつ可能性があるので表内の特定の行を他と区別することができなくなります。重複値をもつカラムを主キーに定義したり、既存の主キーに重複値を入力することはできません。

外部キーは、別の表の主キーまたは一意索引に対応するキーです。二つの表は、共通のキーデータによって親子関係が付けられます。親表は主キーまたは一意索引をもち、子表は外部キーをもちます。

ON UPDATE／ON DELETEキーワードはオプションです。これらのキーワードは、子キーによって参照される親キーの値を更新/削除するときに対応する全ての子キーの行に実行される参照アクションを指定します。参照アクションには、CASCADE、SET NULL、SET DEFAULT、NO ACTIONがあります。

CASCADEは、親キーを更新/削除するときに対応する全ての子キーの値を更新/削除します。親キーを更新するときは、子キーの値を親キーの値と同じにします。親キーを削除するときは、親キーの値と一致する子キーの値を全て削除します。

SET NULLは、親キーを更新/削除するときに対応する子キーの値をNULLに設定します。子キーカラムにNOT NULL制約が定義されているときは、SET NULLアクションを使用することはできません。

SET DEFAULTは、親キーを更新/削除するときに対応する子キーの値をカラムの初期設定値に設定します。子キーカラムにNOT NULL制約が定義されており初期設定値がNULLのときは、SET DEFAULTアクションを使用することはできません。

NO ACTIONは、通常の参照整合性の規則に従います。参照アクションを定義しないと、NO ACTIONが初期設定のアクションになります。

表の外部キーの個数には制限がありません。親キーは表の主キーまたは他の任意の一意索引ですが、子キーを追加する前に親キーを作成しておきます。親キーと子キーのカラム数とカラムタイプまたはカラム長は同じでな

ければなりません。両表の対応カラムの順序は異なっていてもかまいませんが、ALTER TABLE FOREIGN KEYの中では対応する順にリストします。初期設定では親表の主キーを親キーにします。

外部キーカラムにはNULL値があってもかまいません。外部キーがNULL値のときは、自動的に参照整合性が満たされます。ビューに外部キーを作成することはできませんが、シノニムには作成することができます。外部キー名の最大長は32字です。文字、数字、アンダースコア、記号 $ と # を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1**　**Accounts**表の**CustNo**カラムに**Customers**表を参照する外部キー**fkey_CNo**を作成します。親キーのカラム名を指定していないので、**Customers**表の主キーが親キーになります。**Customers**表の主キーは、この実行前に定義されていなければなりません：

```
ALTER TABLE Accounts FOREIGN KEY fkey_CNo (CustNo)
                        REFERENCES Customers;
```

➲ **使用例2**　親キーに**CustNo**カラムを指定して使用例1と同じ外部キー**fkey_CNo**を作成します。**CustNo**カラムは、**Customers**表の主キーか一意索引のどちらかです。この実行前に**Customers**表に主キーか一意索引が定義されていなければなりません：

```
ALTER TABLE Accounts FOREIGN KEY fkey_CNo (CustNo)
                        REFERENCES Customers (CustNo);
```

➲ **使用例3**　表**Invoice**のカラム**PartNo**、**SuppNo**に**Stock**表を参照する外部キー**fkey_No**を作成します。**Invoice**表のカラムの順序 **(PartNo**、**SuppNo)**は**Stock**表の対応カラムの順序**(SuppNo**、**PartNo)**と異なりますが、各表の対応カラムを同じ順序でリストしているので有効なSQL文になります：

```
ALTER TABLE Invoice FOREIGN KEY fkey_No (SuppNo, PartNo)
                        REFERENCES Stock (SuppNo, PartNo);
```

➲ **使用例4**　参照アクションを定義して使用例3と同じ外部キー**fkey_No**を作成します。**ON UPDATE SET DEFAULT**キーワードは、親キーを更新する

ときに対応する子キーカラムの値を初期設定値に設定します。**ON DELETE SET NULL**キーワードは、親キーを削除するときに対応する子キーの値を**NULL**に設定します：

```
ALTER TABLE Invoice FOREIGN KEY fkey_No  (SuppNo, PartNo)
                    REFERENCES Stock (SuppNo, PartNo)
                    ON UPDATE SET DEFAULT
                    ON DELETE SET NULL;
```

関連SQL

ALTER TABLE DROP FOREIGN KEY

ALTER TABLE DROP PRIMARY KEY

ALTER TABLE PRIMARY KEY

CREATE INDEX

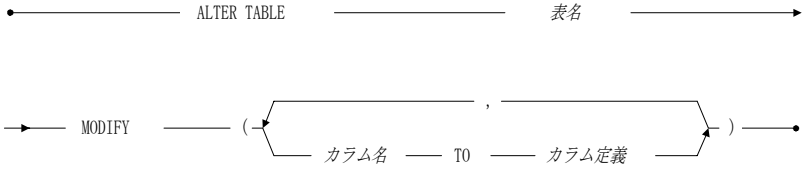
CREATE TABLE

3.15 ALTER TABLE MODIFY COLUMN

目的

表のカラム定義を変更します。

構文



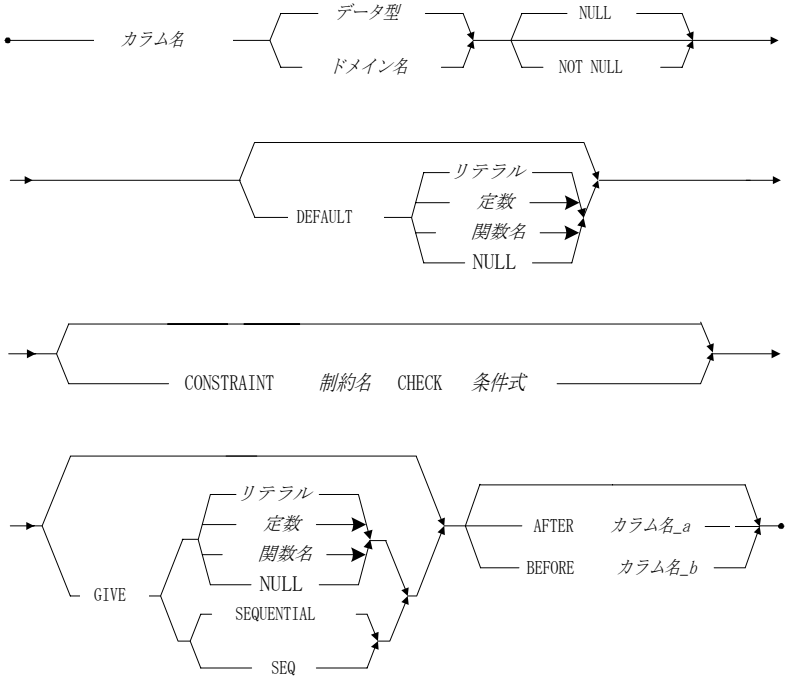
表名	カラム定義を変更する表の名前
カラム名	変更するカラムの名前
カラム定義	カラムの新定義

説明

ALTER TABLE MODIFY COLUMN文は、表の既存カラムの定義を変更します。表所有者、DBA、SYSADM、表のALTER権限をもつユーザーだけがこのSQL文を実行することができます。

カラム定義は、カラム名とデータ型またはドメインで指定します。複数のカラム定義を変更することができます。表の最大カラム数は252です。カラム定義の構文とキーワードの使用法を以下に示します：

カラム定義



カラム名	変更したカラムの名前
データ型	変更したカラムのデータ型
ドメイン名	変更したカラムのドメイン名
リテラル	値が挿入されないときに使用するリテラル値
定数	値が入力されないときに使用する定数値
制約名	カラムに設ける制約の名前
関数名	値が入力されないときに使用する組み込み関数
条件式	評価結果が真または偽となる式
カラム名_a	変更したカラムの直前の既存カラムの名前
カラム名_b	変更したカラムの直後の既存カラムの名前

DBMasterは次のデータ型をサポートします。BINARY、CHAR、DATE、DECIMAL（NUMERIC）、DOUBLE、FILE、FLOAT、INTEGER、LONG VARBINARY（BLOB）、LONG VARCHAR（CLOB）、OID、SERIAL、SMALLINT、TIME、TIMESTAMP、VARCHAR。

データ型の代わりにユーザー定義ドメインを指定することができます。ドメインは、データ型、初期設定値、制約の組み合わせです。ドメインを使用してカラムを定義すると、これらがカラムに適用されます。初期設定値と制約については、以下のDEFAULTとCHECKキーワードの説明を参照してください。カラム定義で初期設定値と制約を指定すると、ドメインの初期設定値と制約が打ち消されます。カラム定義でドメインの制約に制約を追加することができます。

NULL／NOT NULLキーワードはオプションです。これらのキーワードは、カラムがNULL値を取れるか、新しい行を挿入するときにカラムを空のままにできるかを指定します。NULLキーワードは、行を挿入するときにカラム値を未定にしてもよい指定です。NOT NULLキーワードは、行を挿入するときにカラム値を挿入しなければならない指定です。NULLと定義されていたカラムをNOT NULLに変更することは、表が空のときかGIVEキーワードを使用しない限りできません。

DEFAULTキーワードはオプションです。このキーワードは、新しく挿入する行のカラムの値が空のときにカラムに挿入される初期設定値を指定します。初期設定値としては、リテラル値、定数、組み込み関数値、NULLキーワードを指定することができます。組み込み関数は、PI( )、NOW( )、USER( )のように引数の無い関数を使用します。DEFAULT値をNULLにすると、カラムをNOT NULLに定義することはできません。データ型の代わりにユーザー定義のドメインを使用するときは、通常、DEFAULTキーワードは使用しません。DEFAULT句はドメインに含まれるのが普通です。

CHECKキーワードはオプションです。このキーワードは、カラムに入力できる値の範囲を指定します。真または偽と評価される任意の条件式で許容値の範囲を指定します。CHECK条件式の中では、カラムの値を表すのにVALUEキーワードを使用することができます。SQL文がCHECK条件を満たさないときは、そのSQL文は処理されません。データ型の代わりにドメインを使用するときは、通常、CHECKキーワードは使用しません。CHECK句はドメインに含まれるのが普通です。

GIVEキーワードはオプションです。このキーワードは、表の既存の行の変更されたカラムの値がNULLのときに挿入される値を指定します。カラムをNULLからNOT NULLに変更するときは、GIVEキーワードを指定しないとカラムを変更することができません。リテラル値、定数、組み込み関数値、NULLキーワードをGIVE値に指定することができます。GIVE値としてNULLを使用するときは、カラムをNOT NULLに定義することはできません。SERIALデータ型にカラムを変更するときは、GIVEキーワードと共にSEQUENTIAL／SEQキーワードを指定します。SEQUENTIAL／SEQは、SERIALデータ型のカラム定義で指定された値から始まるシリアル番号を既存の行に挿入する指定です。シリアル番号は行が挿入されるごとに増分されます。

BEFORE／AFTERキーワードはオプションです。これらのキーワードは、変更されたカラムの挿入位置を既存カラムとの関係で指定します。BEFOREキーワードは、指定した既存カラムの前（左）に変更カラムを挿入します。AFTERキーワードは、指定した既存カラムの後（右）に変更カラムを挿入します。BEFORE／AFTERキーワードを指定しないと、変更カラムは元の位置のままです。

表のカラムを変更すると、表に定義されたビューとストアド・コマンドは全て不正になります。しかしシノニムには影響を与えません。カラム名の最大長は32字です。文字、数字、アンダースコア、記号 $ と # を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1　Employeesinfo**表の**Phone**カラムのデータ型を**CHAR(20)**に変更します：

```
ALTER TABLE Employeeinfo MODIFY (Phone TO Phone CHAR(20));
```

➲ **使用例2　Employeesinfo**表の**Phone**カラムのデータ型を**CHAR(20)**に変更します。更に**NOT NULL**キーワードを追加し、新規の行の挿入時に**Phone**カラムの値を入力必須にします。**NULL**値をもつ既存の行には、**GIVE**キーワードの値が割当てられます：

```
ALTER TABLE Employeesinfo MODIFY (Phone TO Phone CHAR(20)
                                                NOT NULL
                                                GIVE '000-0000');
```

➲ **使用例3　LineItems**表の**Quantity**と**Amount**カラムのデータ型を**INT**に変更します：

```
ALTER TABLE LineItems MODIFY (Quantity TO Quantity INT,
                                    Amount TO Amount INT);
```

関連SQL

ALTER TABLE ADD COLUMN

ALTER TABLE DROP COLUMN

CREATE TABLE

3.16 ALTER TABLE PRIMARY KEY

目的

表の主キーを作成します。

構文

表名	主キーを作成する表の名前
カラム名	主キーのカラムの名前

説明

ALTER TABLE PRIMARY KEY文は、既存の表に主キーの定義を追加します。ALTER TABLE PRIMARY KEYは、表所有者、DBA、表のALTERとINDEX権限をもつユーザーだけ実行することができます。

キーは、表内の特定の行を識別するのに役立てるカラムまたはカラムの組み合わせです。一意キーは、重複するキー値をもつレコードが無いキーのことです。

主キーは、表内の行を一意的に識別するキーです。主キーが無いと行は重複する値をもつ可能性があるので、表内の特定の行を他と区別することはできなくなります。重複値をもつカラムを主キーに定義したり、既存の主キーに重複値を入力することはできません。

外部キーは、別の表の主キーまたは一意索引に対応するキーです。二つの表は、共通のキーデータによって親子関係が付けられます。親表は主キーまたは一意索引、子表は親表のキーカラムに対応する外部キーカラムをもちます。

参照整合性は、子表の外部キー (子キー) の値が親表の主キーまたは一意索引 (親キー) に対応する値をもつことを保証します。参照整合性は、外部キーによって確立される親子関係を使用して表間に施行されます。DBMasterは、表間の参照整合性制約を外部キーの定義を通して自動的にサポートします。子表にレコードを追加するときは、子キーの値が親キーに存在していなければなりません。 同様に、親表からレコードを削除するときは、先に対応する子キーをもつ全てのレコードを削除しなければなりません。

主キーは、各レコードの主キーが一意であることから表のデータ整合性を保証します。主キーカラムは重複値やNULL値をもたないことを意味するので、主キーカラムはNOT NULL制約を付けて定義します。

表には、主キーを一つだけ設けることができます。この理由で主キーには名前を付けません。DBMasterは、各表にPrimaryKeyと名付けられた一意索引を作成し、主キーを内部的に管理します。主キーの索引は自動的に作成されるので、検索パフォーマンスを上げるために別の主キーカラムの索引を作成する必要はありません。

主キーは、最大32カラム、4000バイトまでのカラムで作成することができます。ビューに主キーを作成することはできませんが、シノニムには作成することができます。シノニムに主キーを作成すると、元の表に主キーが作成されます。

使用例

➲ **使用例**　表**Customers**のカラム**CustNo**に主キーを作成します。**CustNo**カラムは**NOT NULL**制約を付けて定義されるので、そのカラムは全ての値が一意であるか、その表が空でなければなりません：

```
ALTER TABLE Customers PRIMARY KEY (CustNo);
```

関連SQL

ALTER TABLE DROP PRIMARY KEY

ALTER TABLE DROP FOREIGN KEY

ALTER TABLE FOREIGN KEY

CREATE INDEX

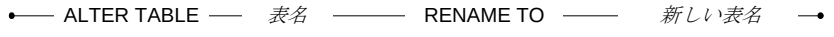
CREATE TABLE

3.17 ALTER TABLE RENAME

目的

既存の表の名称を変更します。

構文



表名	名称を変更する表の旧名称
新しい表名	表の新しい名称

説明

既存の表の名称を変更します。表の所有者、DBA、表のALTER権限を有するユーザーのみがこのSQL文を実行することができます。

表に索引やテキスト索引がある場合は、表名を変更することができます。このSQL文は、ストアド・コマンドやストアド・プロシージャ、トリガー、外部キーのような依存性オブジェクトをサポートしていません。

使用例

➲ **使用例：**

```
ALTER TABLE Customers RENAME TO oldCustomers;
```

関連SQL

CREATE TABLE

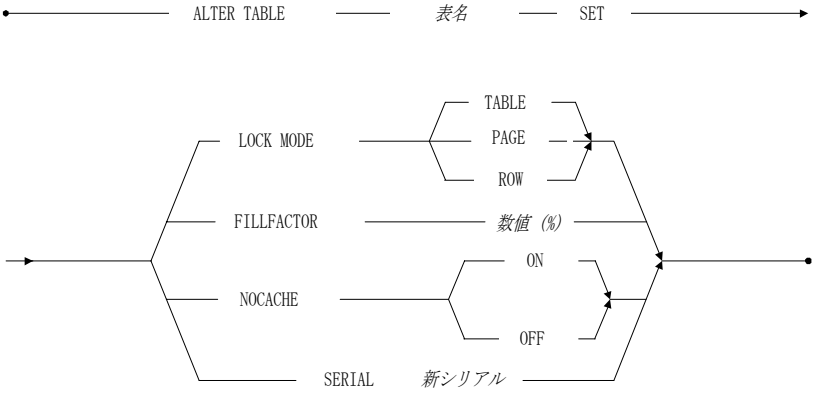
DROP TABLE

3.18 ALTER TABLE SET OPTIONS

目的

表のオプションを設定します。

構文



表名	オプションを変更する表の名前
数値(%)	データページのフィルファクタのパーセント
新シリアル	新しいシリアル番号の開始値

説明

ALTER TABLE SET OPTIONS文は、既存の表のオプション定義を変更します。表の所有者、DBA、表のALTER権限をもつユーザーだけがALTER TABLE SET OPTIONSを実行することができます。

LOCK MODEは、表のデータアクセス時に使用するロックモード（ロックレベル）を指定します。DBMasterには、3つのロックモード、表、ページ、行があります。初期設定はページ・ロックです。表のロック・モードは、SYSTABLEのLOCKMODEカラムを見て判定することができます。

LOCK MODE TABLEは、表全体をロックします。このモードは、ロックされた表の同時並行アクセスを禁止するので同時実行性が低下します。ロックリソースは少なくなり、システム制御域(SCA)の必要メモリも小さくなります。

LOCK MODE PAGEは、データページ単位でロックします。このモードは同時実行性とロックリソースの双方を考慮した選択です。ロックされたページ以外のページは他のユーザーがアクセスできるので中位の同時実行性が得られます。

LOCK MODE ROWは、行単位でロックします。このモードは、ロックされた行以外のデータを他のユーザーがアクセスすることを許すので同時実行性が増大します。ロックリソースの使用は多くなり、SCAの必要メモリも大きくなります。

FILLFACTORは、データページに格納するレコードの最大充填率を指定します。ページ内の既存レコードの更新用スペースを確保しておくことによって、データページ使用の最適化が可能になります。フィルファクタ(充填率)は、50%～100%の範囲内に指定します。表のフィルファクタは、SYSTABLEシステム表のFILLFACTORカラムを見て判定することができます。

NOCACHEは、表検索でデータ・キャッシュに使用するページ・バッファ数を制限します。DBMasterは、直近に使用したページがバッファチェーンの先頭になるようにページを格納します。NOCACHEオプションをONにすると、表検索中に読み込まれたデータページはバッファチェーンの最後に

置かれます。バッファチェーンの最終バッファは表検索前に掃き出され、検索中は後続のデータページが前のページを上書きします。結果として、表検索に使用するページバッファは 1 ページバッファに制限されます。表のキャッシュモードは、SYSTABLEシステム表のCACHEMODEカラムを見て判定することができます。

SERIALオプションは、SERIALカラムのカウンタをリセットします。SERIALカラムには、新しい連続番号が挿入されるようになります。

ALTER TABLE SET OPTIONSは、表に定義されたビューやシノニムには影響しません。

使用例

➲ **使用例1　Customers表のLOCK MODEをTABLEに設定します：**

```
ALTER TABLE Customers SET LOCK MODE TABLE;
```

➲ **使用例2　Customers表のLOCK MODEをPAGEに設定します：**

```
ALTER TABLE Customers SET LOCK MODE PAGE;
```

➲ **使用例3　Customers表のLOCK MODEをROWに設定します：**

```
ALTER TABLE Customers SET LOCK MODE ROW;
```

➲ **使用例4　Customers表のFILLFACTORを90%に設定します：**

```
ALTER TABLE Customers SET FILLFACTOR 90;
```

➲ **使用例5　Customers表のNOCACHEオプションをONにします：**

```
ALTER TABLE Customers SET NOCACHE ON;
```

➲ **使用例6　Customers表のNOCACHEオプションをOFFにします：**

```
ALTER TABLE Customers SET NOCACHE OFF;
```

➲ **使用例7**　表**tb_tmp**の**SERIAL**カウンタを**100**にリセットします：

```
ALTER TABLE tb_tmp SET SERIAL 100;
```

関連SQL

CREATE TABLE

3.19 ALTER TABLE TO ANOTHER TABLESPACE

目的

表、索引を移動します。

構文

表名　　移動された表の名

表領域名　　表を表領域まで移動して、この表領域名

説明

ALTER TABLE TO ANOTHER TABLESPACE コマンドは表を別の表領域まで移動します、同時に表領域に索引があると、この索引も別の表領域まで移動します。DBAの表所有者或いはALTERとINDEX権限を持つ表所有者だけこのコマンドを実行できます。

表を別の表領域に移動することは表をほかのディスクに保存できて、ディスクがフルになることを避けます。

ユーザーはシステム表、一時表或いはビューを別の表領域まで移動することができません。そのほか、標準表をSYSTABLESPACE或いはTMPTABLESPACEまで移動することもできません。

そして、この表は表の索引と同じの表領域に置いていると、ALTER TABLE TO ANOTHER TABLESPACEコマンドを実行してこの索引をほかの表領域まで移動することができます。

使用例

➲ **使用例**

以下の**ts_mode**の表**Employeesinfo** を別の表領域**ts_new**まで移動します。

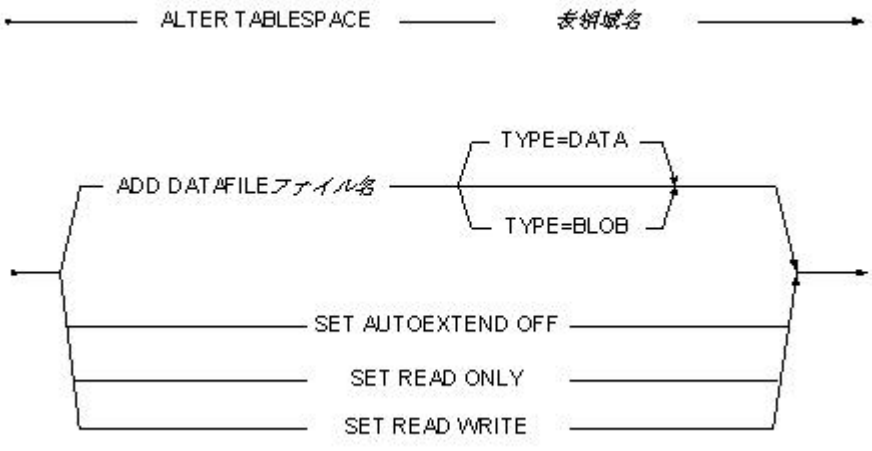
```
ALTER TABLE Employeesinfo MOVE TABLESPACE ts_new;
```

3.20 ALTER TABLESPACE

目的

表領域にファイルを追加します。また、表領域を自動拡張から標準に、あるいは標準から自動拡張に変更します。

構文



表領域名	変更する表領域の名前
ファイル名	表領域に追加するファイルの名前

説明

ALTER TABLESPACEコマンドは既存のテーブルスペースにファイルを追加するか、テーブルスペースのタイプを自動拡張から標準に、または標準から自動拡張に変更するか、テーブルスペースのタイプを読み取り/書き込み可能から読み取り専用に、または読み取り専用から読み取り/書き込み可能に変更します。DBAまたはSYSADMだけがALTER TABLESPACEを実行することができます。

大多数のユーザーにとっては、データがコンピュータに物理的に格納される方法は重要ではありません。DBMasterは、リレーショナル・データモデルを使用して物理ストレージモデルの細部を隠蔽し、代わりに論理ストレージモデルを使用してデータを表現します。

物理ストレージモデルでは、ファイルがデータベースのデータを格納する物理ストレージ構造です。Unixのローデバイスを除いて、ファイルはオペレーティング・システムによって管理され、一方、ファイル内のデータはDBMSによって管理されます。DBMasterは、データ、BLOB、ジャーナルの3種類のファイルを通常の操作に使用します。

ジャーナル・ファイルは、データベースの全ての変更履歴と変更ステータスをリアルタイムに記録する特殊なファイルです。ジャーナルは、失敗したトランザクションによる変更のundo、成功したけれどもデータベース損傷によってディスクに書き込まれていない変更のredoを可能にします。ジャーナル・ファイルは、データベース管理システムのみによって使用され、ユーザーデータの格納には使用されません。

データファイルとBLOBファイルは、ユーザーとシステムのデータを格納します。これらファイルは類似の特性をもっていますが、DBMasterは、パフォーマンスを上げるためにデータとBLOBを別の方法で管理します。データファイルは表と索引のデータを格納し、BLOBファイルはバイナリラージオブジェクト(BLOB)を格納します。

論理ストレージモデルでは、表領域は、データベース情報を管理可能なエリアに区分けするための論理ストレージ構造です。表領域には種々の表や索引が置かれます。表領域内のデータはDBMSによって管理されますが、物理的にはファイルに格納されます。表領域には、標準、自動拡張、システムの3種類があります。

標準表領域には、最低1つのデータファイルあるいはBLOBファイルがあり、サイズが固定しています。標準表領域を拡張するには、既存のファイルを拡大するか新規ファイルを追加します。新規ファイルを追加するときは、**dmconfig.ini**の該当データベース・セクションに、論理ファイル名、物理ファイル名、初期ファイルサイズを指定する行を作成しておきます。標準表領域は最大32767ファイルをもつことができ、ファイルサイズの合計は最大8TBです。Unixプラットフォームでは、ローデバイスに標準表領域を置くことができます。

注：　ローデバイスの詳細情報はUnixシステムのマニュアルを参照してください。

自動拡張表領域は、追加データの必要に応じて自動的にサイズを拡大します。自動拡張表領域には、少なくとも1個のデータファイルがなければなりません。BLOBファイルをもたせることもできます。標準表領域との違いは、自動的に拡張することです。特定の表領域に表を配置することができます。自動拡張表領域にファイルを追加するときは、**dmconfig.ini**の該当するデータベース・セクションに、論理ファイル名、物理ファイル名、初期ファイルサイズを指定する行を作成しておきます。自動拡張表領域はローデバイスをサポートしません。

システム表領域は、データベース作成時に自動的に生成されます。システム表領域はデータベースに1個あり、システムカタログ表が置かれ、スキーマ、セキュリティ、ステータス情報が格納されます。システム表領域は、Unixのローデバイス上に作成する場合を除いて、自動拡張表領域として作成されます。システム表領域には、1個のデータファイルと1個のBLOBファイルが自動的に作成されます。システム表領域は、標準表領域に変換することができます。

SET AUTOEXTEND OFFキーワードは、自動拡張表領域を標準表領域に変更します。表領域が占めるディスク容量を制限するときは、表領域を自動拡張から標準に変更します。

注：　自動拡張表領域のファイルは、8TBを上限にして、ディスクの全使用可能容量まで拡大することがあります。

SET AUTOEXTEND ONキーワードは、標準表領域を自動拡張表領域に変更します。標準表領域を使用し尽くしたときは、表領域を自動拡張に変更することができます。

読み取り専用のテーブルスペースを使用すると、テーブルスペースに変更を加えることはできません。しかし、読み取り専用のテーブルスペースには以下の多数のメリットがあります:

- バックアップの回数が低減されます。読み取り専用のテーブルスペースは読み取り専用に変更後、1度のバックアップを行うだけでよい。
- 復元が容易になります。インスタンスを開始すると、DBMasterメディアのリカバリが不要なのでその点で有効です。
- 読み取り専用のテーブルスペースは更新可能なテーブルスペース(ロックなし) よりも少ないシステムリソースの使用で済みます。

SET READ ONLYのキーワードを使うと、読み取り/書き込み可能なテーブルスペースを読み取り専用のテーブルスペースに変更できます。

SET READ WRITEのキーワードを使うと、読み取り専用のテーブルスペースを読み取り/書き込み可能なテーブルスペースに変更できます。

ADD DATAFILEキーワードは、表領域に新規データファイルまたはBLOBファイルを追加します。表領域に追加するファイルは、同じ物理ディスクではないものにします。Unixではローデバイスにファイルを格納することができます。ローデバイスファイルはオペレーティング・システムのコールを経由せずに書き出すので、通常のファイルよりも高速でアクセスしてパフォーマンスを良くします。

表領域を構成するファイルは、データベース内では論理ファイル名を使用して参照し、物理データとの独立性を維持します。論理ファイル名は、マップ例 1 に示すように**dmconfig.ini**ファイルで、物理ファイル名にマップされます。ディレクトリまたはパスを指定しない新規ファイルは、**dmconfig.ini**のDB_DBDIRキーワードで指定する初期設定のデータベースディレクトリに作成されます。

論理ファイル名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。物理ファイル名の最大長は、ドライブとパス名を含めて255字です。オペレーティング・システムで許される任意の文字、記号を含むことができます。

新規ファイルのタイプは、TYPE=DATAまたはTYPE=BLOBキーワードで指定します。初期設定のファイルタイプはDATAです。

ファイルサイズは、データファイルの場合はページ数、BLOBファイルの場合はBLOBフレーム数で指定します。データページは4KB、8KB、16KB 、32KBで、BLOBフレームは8KB～256KBの範囲で可変です。

*注：　自動拡張表領域の初期サイズは、必要に応じて増分されます。BLOB フレームのサイズは、**dmconfig.ini** ファイルにある DB_BFRSZ キーワードで指定します。*

使用例

➲ **マップ1**　使用例1を実行する前に、論理ファイル名を物理ファイル名にマップし、ページサイズが4KBをもう指定しましたの場合データファイルの初期サイズを4KBのページ数で指定する行を**dmconfig.ini**ファイルに追加します：

```
file1=c:\DBMaster\databases\f1.db 100
```

➲ **使用例1**　表領域**ts_new**に論理ファイル名**file1**をもつ物理ファイル**f1.db**を追加します：

```
ALTER TABLESPACE ts_new ADD DATAFILE file1 TYPE=DATA;
```

- **マップ2**　使用例2を実行する前に、論理ファイル名を物理ファイル名にマップし、初期サイズをフレーム数で指定するエントリを**dmconfig.ini**ファイルに追加します。フレームサイズ8KBを使用すると、**BLOB**ファイルの初期サイズは4000KBになります：

```
file2=c:\DBMaster\databases\f2.bb 500
```

- **使用例2**　表領域**ts_mode**を自動拡張から標準に変更し、次に論理ファイル名**file2**をもつ物理ファイル**f2.bb**を追加します：

```
ALTER TABLESPACE ts_mode SET AUTOEXTEND OFF;

ALTER TABLESPACE ts_mode ADD DATAFILE file2 TYPE=BLOB;
```

- **使用例3**　以下の例はテーブルスペースを読み取り/書き込み可能から読み取り専用に変更します。

```
ALTER TABLESPACE ts_mode SET READ ONLY;
```

関連SQL

CREATE TABLESPACE

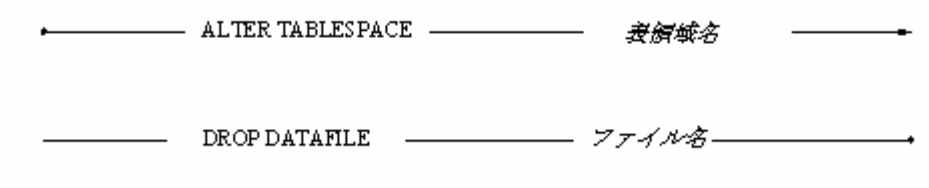
ALTER DATAFILE

3.21 ALTER TABLESPACE DROP DATAFILE

目的

ALTER TABLESPACE DROP DATAFILEコマンドは、表領域から空のデータファイルを削除します。

構文



表領域名..... .. ..データファイルが属する表領域の名前

ファイル名..... 削除するデータファイルの名前

説明

[表領域の修正データファイルの削除]コマンドは、表領域から空のデータファイルを削除します。 DBA または SYSADMのみがコマンドを実行できます。

表領域からデータファイルを削除しているとき、データファイルは空でなければなりません。データファイルにデータが含まれれていると、コマンドが中止され、エラー・メッセージがユーザーに戻されます。 データファイルが表領域に1つしかない場合、データファイルを削除することはできません。 システムの表領域からシステムのデータベースを、または初期設定の表領域から初期設定のデータファイルを削除できないことにも注意する必要があります。

このコメントはロジックファイルだけを削除しますので、ユーザーはこのコメントをコミットした後、手動的で物理ファイルを削除する必要があります。またdmconfig.ini にある情報も削除します。

使用例

➲ **使用例**

表領域ts_newからデータファイルtsfile1を削除する

```
ALTER TABLESPACE ts_new DROP DATAFILE tsfile1
```

3.22 ALTER TRIGGER ENABLE

目的

表の既存トリガーを有効または無効にします。

構文

トリガー名　　有効または無効にするトリガーの名前

表名　　トリガーが定義されている表の名前

説明

ALTER TRIGGER ENABLE文は、表上のトリガーを有効または無効にします。表の所有者、DBA、SYSADMだけがALTER TRIGGER ENABLEを実行することができます。

トリガーは、特定のイベントに応じて事前に定義されたSQL文を自動的に実行するデータベースサーバーの機構です。トリガーによって、標準的なSQL文を使用しては不可能な複雑で非類型的なデータベース操作が実行可能になります。トリガーはデータベース・サーバーの制御下にあるので、ソースの如何に関わらずデータ整合性を保証することができます。トリガーは、ユーザーまたはアプリケーション・プログラムがトリガー・イベントを発生する毎に起動します。

トリガーは、作成されると自動的に有効になります。トリガーを起動させるデータベース操作をテストする等のためにトリガーを無効にするには、DISABLEキーワードを使用します。無効にしてもトリガーはデータベースから削除されません。ENABLEキーワードを用いて再び有効にすることができます。

使用例

➲ **使用例1　Employeesinfo** 表のトリガー**Trig_emp**を無効にします：

```
ALTER TRIGGER Trig_emp ON Employeesinfo DISABLE;
```

➲ **使用例2　Employeesinfo** 表上のトリガー**Trig_emp**を有効にします：

```
ALTER TRIGGER Trig_emp ON Employeesinfo ENABLE;
```

関連SQL

ALTER TRIGGER REPLACE

CREATE TRIGGER

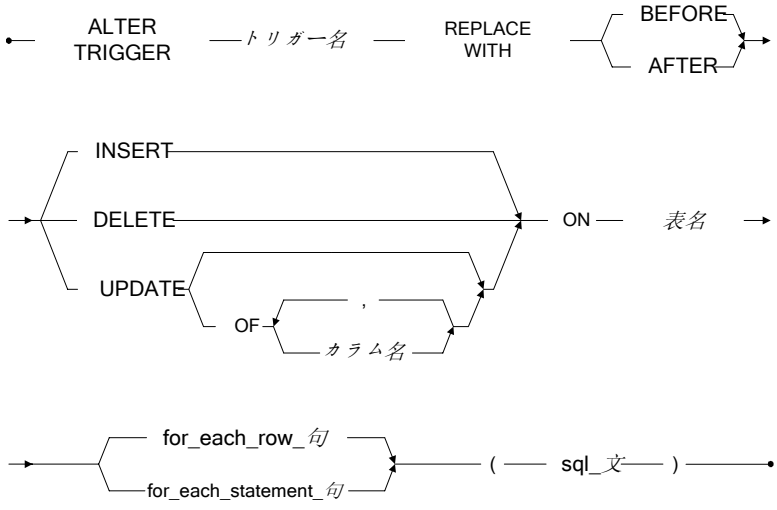
DROP TRIGGER

3.23 ALTER TRIGGER REPLACE

目的

既存のトリガーを新規トリガーで置き換えます。

構文



トリガー名	置き換えられるトリガーの名前
カラム名	新規トリガーを作成するカラムの名前
表名	新規トリガーを作成する表の名前
sql_文	トリガーが起動したときに実行する *SQL* 文

説明

ALTER TRIGGER REPLACE文はトリガーを置き換えます。表所有者、DBA、SYSADMだけがALTER TRIGGER REPLACEを実行することができます。

トリガーは、特定のイベントに応じて事前に定義されたSQL文を自動的に実行するデータベース・サーバーのメカニズムです。トリガーは、標準的なSQL文を使用しては不可能な複雑で非類型的なデータベース操作を実行可能にします。トリガーはデータベース・サーバーの制御下にあり、トリ

ガー起動の原因に関わらずデータ整合性を保証します。ユーザーやアプリケーション・プログラムがイベントを発生する毎に、必ずトリガーが起動します。

トリガーを置き換えるには、トリガー名と新規トリガーのアクション、アクションタイム、イベント、表、タイプを指定します。

注：　ALTER TRIGGER REPLACEは、変更元のトリガー表にのみ働きます。

トリガーは、一般のデータベース・オブジェクトとは異なり、完全修飾名では識別しません。代わりに、トリガーは表に関連付けられます。同じ表の全てのトリガー名は、一意でなければなりません。トリガー・アクションは、イベントを発生するユーザーの権限ではなく、トリガー表の所有者のセキュリティ権限とオブジェクト権限で動作します。

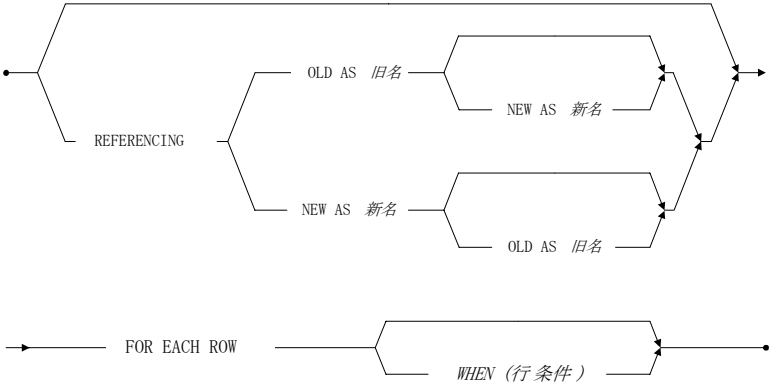
BEFORE／AFTERキーワードは、トリガー・アクションの実行時点をイベントの前／後の関係で指定します。BEFOREキーワードは、トリガー・イベントの前にトリガー・アクションを実行することを意味します。AFTERキーワードは、トリガー・イベントの後にトリガー・アクションを実行することを意味します。

INSERT／DELETE／UPDATEキーワードは、トリガーを引き起こすイベントを指定します。INSERT／DELETEとUPDATEは使用法が少し違います。INSERTキーワードは表に行を挿入する時のトリガー起動を指示し、DELETEキーワードは表から行を削除する時のトリガー起動を指示します。UPDATEキーワードは、表の任意のカラムを更新する時のトリガー起動を指定します。特定のカラムを更新する時のトリガー起動を指示するには、UPDATE OFを使用してカラムリストを指定します。

注：　二つ以上のUPDATE OFトリガーで同じカラム名を使用することはできません。

ONキーワードは、トリガーが定義される表名を指定します。トリガー表は標準表でなければなりません。一時表、ビュー、シノニムにトリガーを作成することはできません。

For Each Row 句



旧名	トリガー・アクション起動前のトリガー表の値を参照する別名
新名	トリガー・アクション起動後のトリガー表の値を参照する別名
行条件	トリガーを起動させる行の条件

REFERENCINGキーワードは、OLDとNEWの別名を指定します。行トリガーは、トリガー・アクションの中でトリガー起動の前／後のカラムの値をOLD／NEWキーワードで参照しますが、同名の表OLDやNEWが既に存在するときには、REFERENCINGで付けた別名を使用します。

FOR EACH ROWキーワードは、トリガー・イベントが行を変更する毎にトリガーを起動する指令です。FOR EACH ROWキーワードを使用して定義された行トリガーは、トリガーを起動させる文が何も行を処理しなければ起動しません。

WHENキーワードは、トリガーを起動させる行が満たすべき条件を指定します。WHEN条件は、トリガー・イベントが行を変更する毎に評価されます。条件が真ならばトリガーが起動し、偽ならば起動しません。

注：　WHEN条件の結果はトリガー・アクションの実行にだけ有効です。トリガーを起動させる文には何の影響も与えません。

For Each Statement 句

FOR EACH STATEMENTキーワードは、トリガーを起動するSQL文に1回だけ起動するトリガーの指定です。FOR EACH STATEMENTキーワードを使用して定義された文トリガーは、トリガーを起動する文が何も行を処理しなくても起動します。

トリガーが起動したときに実行する文のことをトリガー・アクションといいます。トリガー・アクションにすることができるのは、INSERT、UPDATE、DELETE、EXECUTE PROCEDURE文です。トリガー・アクションで組み込み関数を使用するときは、PI( )、NOW( )、USER( )のように引数の無いものだけを使用します。トリガーが呼び出し実行するストアド・プロシージャは、COMMIT、ROLLBACK、SAVEPOINTのようなトランザクション制御文を含むことはできません。

トリガー表の各イベントには、アクションのタイムBEFORE／AFTERとトリガータイプFOR EACH ROW／FOR EACH STATEMENTを組み合わせた複数のトリガーを作成することができます。例えばINSERTイベントにアクション・タイムとトリガータイプを組み合わせて4個のトリガー、BEFORE／FOR EACH STATEMENT、BEFORE／FOR EACH ROW、AFTER／FOR EACH ROW、AFTER／FOR EACH STATEMENTを作成することができます。

注：　同様にUPDATE、DELETE イベントにも複数のトリガーを作成することができます。

UPDATE OFを使用すると、各カラムにアクション・タイム／トリガータイプを組み合わせたトリガーを作成することができます。各カラムに4組のUPDATE OFトリガー、BEFORE／FOR EACH STATEMENT、BEFORE／

FOR EACH ROW、AFTER／FOR EACH ROW、AFTER／FOR EACH STATEMENTを作成することができます。UPDATE OFトリガーが表にあるときは、同じ表にUPDATEトリガーを作成することはできません。また、別のUPDATE OFトリガーで既に使用されているカラムを指定することもできません。

使用例

➲ **使用例1**　**Employeessinfo**表の**Trig_emp**トリガーを変更します。元は**FOR EACH STATEMENT**になっていたトリガーを**FOR EACH ROW**で置き換えます：

```
ALTER TRIGGER Trig_emp REPLACE WITH
                    BEFORE UPDATE ON Employeessinfo
                    FOR EACH ROW
           (INSERT INTO NameChange VALUES (OLD.FName, OLD.LName,
                                           NEW.FName, NEW.LName);
```

➲ **使用例2**　使用例1の**Employeesinfo**表の**Trig_emp**トリガーを変更します。**BEFORE UPDATE** のトリガーイベントを **AFTER INSERT** で置き換えます：

```
ALTER TRIGGER Trig_emp REPLACE WITH
                    AFTER INSERT ON Employeesinfo
                    FOR EACH ROW
           (INSERT INTO NameChange VALUES (OLD.FName, OLD.LName,
                                           NEW.FName, NEW.LName);
```

➲ **使用例3**　使用例2の**Employeesinfo**表の**Trig_emp**トリガーを変更します。**INSERT** のトリガー・アクションを**EXECUTE PROCEDURE** で置き換えます：

```
ALTER TRIGGER Trig_emp REPLACE WITH
                    AFTER INSERT ON Employeesinfo
                    FOR EACH ROW
           (EXECUTE PROCEDURE LogTime);
```

関連SQL

ALTER TRIGGER ENABLE

CREATE TRIGGER

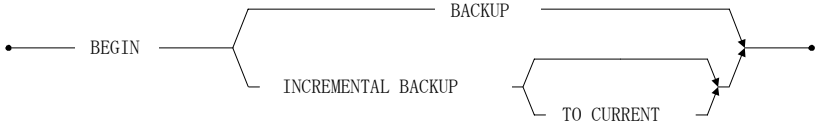
DROP TRIGGER

3.24 BEGIN BACKUP

目的

オンライン・バックアップを開始します。

構文



説明

BEGIN BACKUP文は、ユーザーを切断したり、データベースを終了せずに、データベースの全てのファイルをバックアップできる状態にします。DBAまたはSYSADMだけがBEGIN BACKUPを実行することができます。

メディア障害は、コンピュータ・システムのオンライン補助記憶装置の障害です。最も多く使用される補助記憶装置はハードディスクです。メディア障害は、通常ディスク本体の物理的障害、つまりヘッドクラッシュ、火災、地震、強い振動、物理的な操作限界を超えた加重等によって発生します。

メディア障害が発生すると、いくつかのファイルが物理的に傷つけられます。アーカイブまたはバックアップを用意し、データベースを正しくリストアできるようにしておきます。メディア障害のデータベースをリストアするために、データベース・ファイルを定期的にバックアップしておきます。バックアップには、いくつかの種類があります。

オンライン・バックアップは、データベースの運用中にバックアップを取ります。データベース管理者はデータベースを終了する必要がなく、ユーザーはデータベースを切断する必要がありません。オンライン・バックアップは、ユーザーは何もする必要が無いので非常に便利です。DBMSには、オンラインでデータベースをバックアップする機能が必要です。

オフライン・バックアップは、データベースが終了された後にバックアップを取ります。データベース管理者は、データベースの終了予定時刻を全てのユーザーに通知し、終了前にデータベースから切断するようにします。このため、オフライン・バックアップはユーザーには不便です。ユーザーは、トランザクションを全て終了し、データベースを切断することを認識する必要があります。オフラインの場合、DBMSにデータベースをバックアップする機能は必要ありません。

完全バックアップは、全てのデータとジャーナルのコピー、つまりデータベース全体のコピーを取ります。完全バックアップは、データベース全体をバックアップするので大量のストレージ容量を必要としますが、データベースを素早くリストアすることができます。

差分バックアップは最新の完全バックアップのデータを基礎とします。つまり差分のベースです。差分バックアップは差分ベースを変更されたデータのみ含みます。普通に、差分ベースは連続な差分バックアップに使用されます。リストアタイムで完全バックアップと差分バックアップはこの完全バックアップを基づいて完全データベースを生成します。

増分バックアップは、前回の完全バックアップ以降に変更されたジャーナル・ファイルのコピーだけを取ります。バックアップ・ファイルには、前回の完全バックアップ以降にデータベースに加えられた変更のコピーが取られます。増分バックアップは、ジャーナル・ファイルだけをバックアップするので少量のストレージ容量で済みますが、リストアするのに時間がかかります。

DBMasterは、オフライン完全バックアップ、オンライン完全バックアップ、オンライン増分バックアップ、オンライン差分バックアップ、現在までのオンライン増分バックアップの5種類のバックアップをサポートします。増分バックアップを取るときは、事前にオフライン完全バックアップまたはオンライン完全バックアップのどちらかを取っておきます。完全バックアップを取っておかないと、メディア障害のデータベースをリストアできないかもしれません。

オフライン完全バックアップを取るときは、全てのユーザーが切断しているのを確認してからデータベースを終了します。終了中にエラーが発生すると、バックアップ操作が完了しなかったり、データベースをリストアできないかもしれません。全てのデータ、BLOB、ジャーナルのファイルをバックアップします。オフライン完全バックアップを使用すると、終了時点のデータベースにリストアすることができます。

オンライン完全バックアップを取るときは、NON-BACKUP、BACKUP-DATA、BACKUP-DATA-AND-BLOBモードでデータベースを起動しておきます。バックアップを開始するには、BEGIN BACKUPを実行します。全てのデータファイルとBLOBファイルをバックアップした後、END BACKUP DATAFILEを実行します。次に全てのジャーナル・ファイルのバックアップを取ります。その後で、END BACKUP JOURNAL文を実行してバックアップを完了し、データベースを通常の操作に戻します。オンライン完全バックアップを使用すると、END BACKUP DATAFILEの実行時点からアクティブ・ジャーナル・ファイルをコピーした時点までのデータベースにリストアすることができます。

差分バックアップは、NON-BACKUP、BACKUP-DATA、またはBACKUP-DATA-AND-BLOB モードでデータベースを起動します。差分バックアップを作成する前、差分のベース(完全バックアップ)が存在することが確保しなければなりません。差分バックアップはデータファイル(DB とBB)だけをコピーしますが、ジャーナルではありません。ジャーナルファイルは頻繁に変更されるからです。それで、差分バックアップを実行する場合、有効なジャーナルブロックだけはコピーされます。

オンライン増分バックアップまたは既存なオンライン増分バックアップを取るときは、BACKUP-DATAかBACKUP-DATA-AND-BLOBモードでデータベースを起動しておきます。

オフライン完全バックアップは、データベース・ファイルのオペレーティング・システム読み込み許可をもつユーザーだけが実行することができます。オンライン・バックアップは、DBAかSYSADMセキュリティ権限をもつユーザーだけが実行することができます。オンライン・バックアップは、一度に1ユーザーだけが実行することができます。

オンライン・バックアップを中止するには、ABORT BACKUPを実行します。中止されたバックアップのファイルを使用してデータベースをリストアすることはできません。

オンライン完全バックアップとオンライン差分バックアップは、いつでも任意のバックアップ・モード(NON-BACKUPを含む)でバックアップを取ることができます。オンライン増分バックアップは、BACKUP-DATAかBACKUP-DATA-AND-BLOBモードでデータベースが作動中のときだけ取ることができます。

バックアップ・モードは、オンライン増分バックアップでバックアップする情報のタイプを指示します。バックアップ・モードの設定は、**dmconfig.ini**ファイルのDB_BMODEキーワード（オフライン時）、**dmSQL**の**SET**文（オンライン時）、JConfiguration Tool（オフライン時）、JServer Manager「データベース起動の高度な設定」ダイアログ（オフライン時）、JServer Manager「ランタイムの設定」ダイアログ（オンライン時）を使用して変更します。

NON-BACKUPモードは、前回の完全バックアップ以降に挿入/更新した全てのデータを保護しません。このモードでは、オンライン増分バックアップを取ることはできません。データベースのプログラム障害はジャーナルを使用して完全にリカバリされますが、メディア障害はデータ紛失を起こすかもしれません。アクティブ・トランザクションが使用していないジャーナル・ブロックは、チェックポイント後、直ちに再利用されます。しかし、ジャーナル・ブロックが上書きされると、データベースは前回の完全バックアップの時点にリストアすることができるだけになります。

DB_BMODEキーワードを使用してバックアップ・モードをNON-BACKUPにセットするには、テキストエディタを使用して**dmconfig.ini**ファイルを開き、DB_BMODEの値を0にします。オンライン完全バックアップをとるときにNON-BACKUPにセットするには、SET BACKUP OFF文を使用します。SET BACKUP OFF文は、オンライン完全バックアップのBEGIN BACKUPの後、END BACKUP JOURNALの前に実行しなければなりません。

BACKUP-DATAモードは、前回の完全バックアップ以降に挿入/更新されたBLOB以外のデータを保護します。オンライン増分バックアップを取ることができますが、BLOBデータはジャーナルに記録されないのでバックアップ・ジャーナル・ファイルにも保存されません。前回の完全バックアップ以降に挿入/更新されたレコードにあるBLOBデータは、NULL値に置き換えられます。データベースをリストアした後に、新しいBLOBデータをもつ全てのレコードを手作業で更新します。データベースのプログラム障害はジャーナルを使用して完全にリカバリされ、ディスク障害は部分的にリカバリされます。

DB_BMODEキーワードを使用してバックアップ・モードをBACKUP-DATAにセットするには、テキストエディタを使用して**dmconfig.ini**ファイルを開き、DB_BMODEの値を1にします。オンライン完全バックアップ中にBACKUP-DATAにセットするには、SET BACKUP ON文を使用します。SET BACKUP ON文は、オンライン完全バックアップのBEGIN BACKUPの後、END BACKUP JOURNALの前に実行しなければなりません。

BACKUP-DATA-AND-BLOBモードは、前回の完全バックアップ以降に挿入/更新されたBLOBを含む全てのデータを保護します。オンライン増分バックアップを取ることができ、全てのデータがバックアップ・ジャーナル・ファイルに保存されます。データベースのプログラム障害はジャーナルを使用して完全にリカバリされ、ディスク障害も完全にリカバリされます。最新のバックアップを使用して、メディア障害の時点のデータベースにBLOBデータを含めて完全にリストアします。アクティブ・トランザクションが使用していないジャーナル・ブロックは、チェックポイント後、且つ、ジャーナル・ファイルがバックアップされた後に再利用されます。

DB_BMODEキーワードを使用してBACKUP-DATA-AND-BLOBバックアップ・モードにセットするには、テキストエディタを使用して**dmconfig.ini**ファイルを開き、DB_BMODEの値を2にします。オンライン完全バックアップ中にBACKUP-DATA-AND-BLOBにセットするには、SET BLOB BACKUP ON文を使用します。SET BLOB BACKUP ON文は、オンライン完全バックアップのBEGIN BACKUPの後、END BACKUP JOURNALの前に実行しなければなりません。

使用例

➲ **使用例1**　オンライン完全バックアップを取る手順を示します。はじめに**BEGIN BACKUP**を実行して完全バックアップを取ることを**DBMaster**に知らせます。次に**OS**コマンドを使用して全てのデータファイルと**BLOB**ファイルをバックアップ場所にコピーし、**END BACKUP DATAFILE**を実行します。次に**OS**コマンドを使用して全てのジャーナル・ファイルをバックアップ場所にコピーし、**END BACKUP JOURNAL**を実行します。データベースは通常の操作に戻ります：

```
BEGIN BACKUP;
    Copy data and BLOB files to backup location using OS commands
    Change backup mode if desired
    Abort the backup if desired
END BACKUP DATAFILE;
    Copy Journal files to backup location using OS commands
    Change the backup mode if desired
    Abort the backup if desired
END BACKUP JOURNAL;
```

関連SQL

ABORT BACKUP

END BACKUP

SET CONNECT OPTIONS

3.25 BEGIN WORK

目的

トランザクションの開始を表示します。

構文

説明

BEGIN WORK文はオプションです。DBMasterはこれを無視します。スクリプトファイルでトランザクションの開始を文書化するのに使用します。

使用例

➲ **使用例**　トランザクションの開始を文書化する**BEGIN WORK**を例示します。テキストはスクリプトファイルの何処にあってもかまいません：

```
BEGIN WORK;
    ...
    SQL文
    SQL文
    ...
COMMIT WORK;
```

関連SQL

COMMIT WORK

ROLLBACK

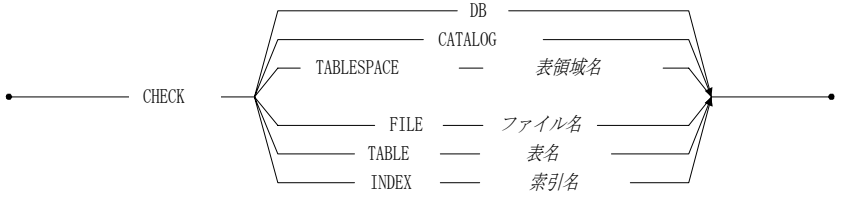
SAVEPOINT

3.26 CHECK

目的

データベースオブジェクトのデータ整合性をチェックします。

構文



表領域名	チェックする表領域の名前
ファイル名	チェックするファイルの名前
表名	チェックする表の名前
索引名	チェックする索引の名前

説明

CHECK文は、指定したデータベースオブジェクトのデータ整合性をチェックします。問合せ結果に不整合や誤りがあったり、エラーメッセージが頻発したり、見慣れないエラーメッセージが返されるときには、データベースの整合性をチェックします。オブジェクトの所有者、DBA、SYSADMだけがCHECKを実行することができます。

データベース、索引、表、ファイル、表領域、システムカタログの整合性をチェックすることができます。整合性チェックは時間とリソースを消費します。OFFピーク時に必要なものだけのCHECKを実行するようにし、ユーザーの不便を最小限にします。

データベースオブジェクトのチェックは、システムカタログ表の情報が正しく有効であることの確認チェックから始まります。システムカタログにエラーがある場合は、データベースに深刻な整合性エラーがあり直ちにチェックを終了します。次に、オブジェクトと関連オブジェクトの物理構造とデータ一貫性をチェックします。オブジェクトをチェックするときは、オブジェクトに含まれる関連オブジェクトも全てチェックします。索引、データページ、ファイル、表もチェックします。

ある種のエラーは修復することができます。通常、索引を削除し再作成することによって、多くの問題を修復することができます。破壊された表は、表の全レコードをアンロードし、表を削除して再作成し、全データをロードすることによって修復できる可能性があります。

データベースに整合性エラーがある場合は、直ちに全データとジャーナル・ファイルのバックアップを取ります。クラッシュ後のリカバリによって、ある種の整合性エラーが直されることがあります。クラッシュリカバリは、データベースを終了し再起動すると働きます。データベースが再起動した後に、CHECKを再実行してエラーが直されているか調べます。

依然として不整合があるときは、CASEMakerカスタマーサービスに連絡してください。CASEMakerカスタマーサポートのスタッフがデータベース修復の手助けをします。

注：　CASEMakerカスタマーサービスへの連絡方法については、ライセンス契約書をご覧ください。

使用例

➲ **使用例1　Customers**表のデータ整合性をチェックします：

```
CHECK TABLE Customers;
```

➲ **使用例2　Customers**表の索引**idxCustNum**のデータ整合性をチェックします；索引名を指定するときは表名も指定します：

```
CHECK INDEX Customers.idxCustNum;
```

- **使用例3**　**customer_data**ファイルにあるデータページまたは**BLOB**フレームのデータ整合性をチェックします：

```
CHECK FILE customer_data;
```

- **使用例4**　表領域**ts_new**にある全てのファイル、表、データページ、表データの整合性をチェックします：

```
CHECK TABLESPACE ts_new;
```

- **使用例5**　データベースのシステムカタログの整合性をチェックします：

```
CHECK CATALOG;
```

- **使用例6**　全てのデータベース・オブジェクトの整合性をチェックします：

```
CHECK DB;
```

関連SQL

CREATE INDEX

CREATE TABLE

CREATE TABLESPACE

CREATE TEXT INDEX

3.27 CHECKPOINT

目的

強制的にチェックポイントを取ります。

構文

説明

CHECKPOINT文は、強制的にチェックポイントを取ります。データベースが頻繁にアクセスされており、バックアップを取ったり、データベースを再起動することができないときにはチェックポイントを取ります。DBAとSYSADMだけがCHECKPOINTを実行することができます。

チェックポイントは、データベースの状態をクリーンにします。全てのジャーナルレコードと使用データページをメモリバッファからディスクに書き出し、バックアップやリカバリに不要になったジャーナル・ブロックを再利用します。最も古いアクティブ・トランザクションが起動する前に完了している非アクティブ・トランザクションのジャーナル・ブロックが再利用されます。

チェックポイントを取ると、プログラム障害後の起動タイムが短縮されます。DBMasterは、最終チェックポイント時刻とチェックポイント時に活動中のトランザクションリストをジャーナル・ファイルのヘッダに書き出します。この情報を利用して、データベースのリカバリ時にundo／redo／無視するトランザクションが決定されます。

チェックポイントは、データベースの起動時と終了時、オンライン・バックアップを取るとき、ジャーナルフルのときに自動的に取られます。チェックポイントを取るには、最終チェックポイント以降のトランザクション数とサイズによっては大量の時間がかかるかもしれません。トランザクションは、チェックポイントを取っている間、待たされます。ジャーナルフルの場合、チェックポイントを取ってもトランザクションを完了させるのに必要なジャーナル・ブロックを再生することができないときは、トランザクションは中止されます。この場合は、中止されたトランザクションの全ての命令文を再実行します。

トランザクション処理中の遅延を避けるために、CHECKPOINTを使用して定期的に手動チェックポイントを取ります。定期的な手動チェックポイントは、データベースの起動・終了・バックアップの時間、チェックポイントが完了するまでのトランザクションの待ち時間、ジャーナルフルの可能性を減らすことができます。手動チェックポイントの最適間隔は、データベースのアクティビティ頻度に依存します。

使用例

➲ **使用例**　強制的にチェックポイントを取ります：

```
CHECKPOINT;
```

関連SQL

BEGIN WORK

COMMIT WORK

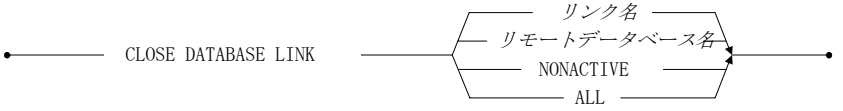
ROLLBACK

3.28 CLOSE DATABASE LINK

目的

リモートデータベースへのリンクをクローズします。

構文



リンク名	クローズするリモート・データベース・リンクの名前

リモート・データベース名　　全てのリンクをクローズするリモート・データベースの名前

説明

CLOSE DATABASE LINK文は、リモート・データベースへのリンクをクローズします。単一リンクのクローズ、複数リンクの同時クローズに使用します。リモート・データベースに接続中のリンクをもつユーザーは誰でもCLOSE DATABASE LINKを実行することができます。

データベース・リンクは、リモート・データベースと接続し、ローカル・データベースからリモート・データベースへのアクセスを可能にします。リンクにはセキュリティ情報が追加されます。リンクは、別のユーザー名でリモートデータベースに接続することを可能にします。アカウントのないリモートデータベースに接続するには、パブリックリンクを使用します。

リンク名を指定してCLOSE DATABASE LINKを実行すると、アクティブ・トランザクションがないリモート・データベースへのリンクの場合、リンクをクローズします。リモート・データベースを指定してCLOSE DATABASE LINKを実行すると、リモート・データベースに接続する全てのリンクをクローズします。

注：　リンクがアクティブ・トランザクションをもつときはエラーが返され、リンクはオープンのまま残ります。トランザクションの終了を待ってから、再度、リンクのクローズを試みます。

NONACTIVEキーワードは、アクティブなトランザクションによって使用されていない全てのリモート・データベースへのリンクをクローズします。トランザクションが使用中のリンクはオープンのまま残ります。リンクをクローズするには、トランザクション終了まで待ってから再度クローズします。

ALLキーワードは、全てのリモート・データベースへのリンクをクローズします。トランザクションが使用中のリンクがあると、リンクをオープンのままにしてエラーを返します。リンクをクローズするには、トランザクション終了まで待ちます。

使用例

- **使用例1　FieldLink**リンクをクローズします：

```
CLOSE DATABASE LINK FieldLink;
```

- **使用例2**　ローカル**dmconfig.ini**ファイルにあるリモート・データベース名**FieldOffice**への全てのリンクをクローズします：

```
CLOSE DATABASE LINK FieldOffice;
```

- **使用例3**　アクティブ・トランザクションで使用されていない全てのリンクをクローズします：

```
CLOSE DATABASE LINK NONACTIVE;
```

- **使用例4**　アクティブ・トランザクションによって使用されていなければ、全てのリンクをクローズします。クローズしない場合は、エラーが返されリンクはオープンのままになります：

```
CLOSE DATABASE LINK ALL;
```

関連SQL

CREATE DATABASE LINK

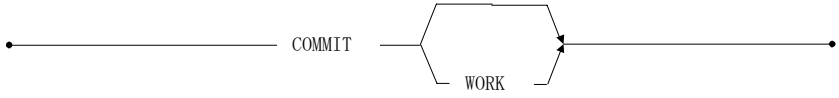
DROP DATABASE LINK

3.29 COMMIT WORK

目的

現在のトランザクションをコミットします。

構文



説明

COMMIT WORK文はトランザクションをコミットします。COMMIT WORKを実行すると、新トランザクションが自動的に起動します。CONNECT以上のセキュリティ権限をもつユーザーは、誰でもCOMMIT WORKを実行することができます。

トランザクションは、データベースの整合性を維持するための論理作業単位であり、伝統的に一括して完了する必要がある一連のデータベース操作と定義されています。トランザクションは統合体であり、成功してデータを変更するか、失敗してデータを変更しないかのいずれかになります。

例えば、出荷レコードと在庫レコードの2種類の情報がデータベースに格納されており、共に製品の数量をもっているとします。製品を出荷するときには、出荷リストに製品と数量が追加されます。出荷された数量は在庫から減らさなければなりません。両方の作業が共に完了しなければ、データベースは不整合の状態になります。出荷されたのに在庫から減らされなければ製品の在庫数量が多くなり過ぎ、在庫から減らされたのに出荷されなければ在庫数量が少なくなり過ぎます。出荷と在庫の二つのオペレーションは、一つのトランザクションとして共に成功するか、そうでなければ両方とも失敗にしなければなりません。

トランザクションがコミットされると、データは変更されトランザクションは成功して終了します。トランザクションがロールバックされると、データは変更されずトランザクションは失敗します。

COMMIT WORKを実行すると、現在のトランザクションによる全ての変更がデータベースに書き出されます。COMMIT WORKはトランザクションの

変更データを書き出すだけです。データベース接続がAUTOCOMMIT モードのときは、COMMIT WORKは不要になります。

AUTOCOMMITモードは、トランザクションのコミットを制御します。AUTOCOMMITモードがONのときは、各SQL文が一つのトランザクションとして取り扱われます。SQL文を実行し成功して終了する場合、自動的にSQL文をコミットし、実行エラーが発生すればSQL文をロールバックします。AUTOCOMMITモードがOFFのときは、二つのCOMMIT WORKの間の全てのSQL文が一つのトランザクションを形成します。トランザクション内の変更は、COMMIT WORKを実行するとコミットされ、ROLLBACK WORKを実行するとロールバックされます。

データベースがクラッシュした場合、コミットされなかったトランザクションは全てロールバックされます。ロールバックされたトランザクション内の変更をデータベースに反映するには、データベースを再起動したときに、トランザクション内の全てのSQL文を再実行します。

使用例

➲ **使用例　AUTOCOMMIT**モードをOFFにすると、二つの**COMMIT WORK**の間で実行された全ての**SQL**文による変更が2番目の**COMMIT WORK**でコミットされます：

```
COMMIT WORK;
    ...
    SQL Command
    SQL Command
    ...
COMMIT WORK;
```

関連SQL

BEGIN WORK

ROLLBACK

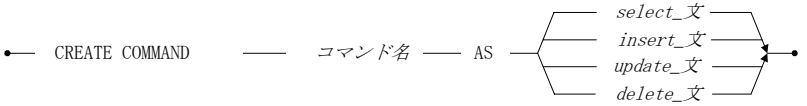
SAVEPOINT

3.30 CREATE COMMAND

目的

ストアド・コマンドを新規に作成します。

構文



OR REPLACE	OR REPLACEで既存なストアドコマンドを再び作成します。この句を使用して既存なストアドコマンドの定義を変更できます
コマンド名	新規に作成するストアド・コマンドの名前
select_文	有効なSELECT文
insert_文	有効なINSERT文
update_文	有効なUPDATE文
delete_文	有効なDELETE 文

説明

CREATE COMMAND文は、ストアド・コマンドを新規に作成します。ストアド・コマンドは、頻繁に使用するSQLデータ操作文を簡便に素早く実行するのに使用します。CREATE COMMANDは、RESOURCE以上のセキュリティ権限をもち、使用するSQL文を実行するのに必要な全てのセキュリティ権限とオブジェクト権限をもつユーザーだけが実行することができます。

ストアド・コマンドは、コンパイルされた実行形式のSQLデータ操作文で永続的にデータベースに格納されます。ストアド・コマンドは、コンパイル・最適化せずにSQLデータ操作文を繰り返し実行できるようにします。ストアド・コマンドはストアド・プロシージャと似ていますが、一つのSQLデータ操作文だけからなり、プログラムロジックをもたない点が異なります。

ストアド・コマンドを作成するには、コマンド名と有効なSQLデータ操作文SELECT、INSERT、UPDATE、DELETEを指定します。データ操作文のカラム値の位置にホスト変数を使用して、コマンド実行時に実際のカラム値を割当てることができます。ホスト変数を使用するには、データあるいはカラムの値を疑問符(?)で置き換えます。

ホスト変数をもつストアド・コマンドは、組み込み関数の値、NULLキーワード、DEFAULTキーワード、他のホスト変数の値を実行時に与えます。組み込み関数は、RAND( )、PI( )、CURDATE( )、NOW( )のように引数の無いものだけを使用します。NULL値を与えるときは、ホスト変数のカラムがNULL値を受け入れるものでなければなりません。ストアド・コマンド実行時に与えるパラメータの個数は、ストアド・コマンド定義時のホスト変数の個数と一致しなければなりません。

ストアド・コマンドは、参照する表やカラムが削除されたり、BEFOREやAFTERキーワードを使用してカラム定義が変更されると、不正になり使用できなくなります。BEFOREおよびAFTERキーワードを使用しないで表にカラムを追加した場合は、ストアド・コマンドは何の影響も受けません。不正なストアド・コマンドは、削除してデータベースから削除します。

ストアド・コマンド名は、データベースで一意でなければなりません。ストアド・コマンド名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1　Employeesinfo** 表からラストネームが ‘**A**’ で始まる全ての従業員を検索するストアド・コマンド**sc_select** を作成します：

```
CREATE COMMAND sc_select AS SELECT * FROM Employeesinfo WHERE LastName LIKE
‘A%’;
```

➲ **使用例2　Employeessinfo** 表の **Manager** カラムの値をホスト変数を使用して更新するストアド・コマンド**sc_update** を作成するため、以下の構文を使用します：

```
CREATE COMMAND sc_update AS UPDATE Employeessinfo SET Manager = ? WHERE
Manager = ?;
```

或いは：

```
CREATE COMMAND OR REPLACE sc_update AS UPDATE Employeesinfo SET Manager = ?
WHERE Manager = ? ;
```

関連SQL

DROP COMMAND

EXECUTE COMMAND

GRANT(実行権限)

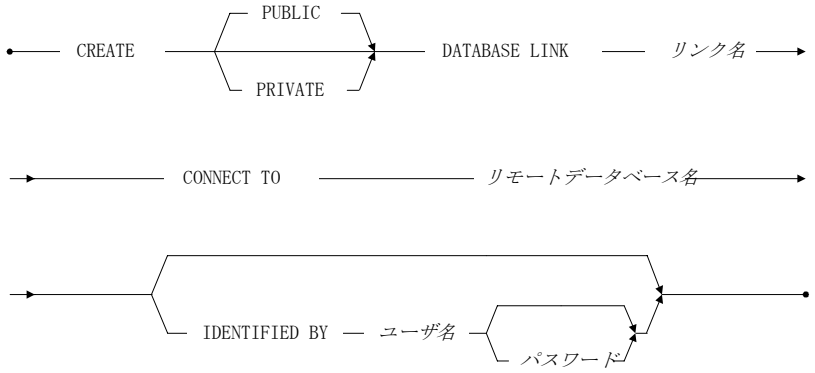
REVOKE (実行権限)

3.31 CREATE DATABASE LINK

目的

リモート・データベースへのリンクを新規に作成します。

構文



リンク名	作成するリンクの名前
リモート・データベース名	接続するリモート・データベースの名前
ユーザー名	リモート・データベースのユーザーの名前
パスワード ード	リモート・データベースユーザーのパスワ

説明

CREATE DATABASE LINK文は、リモート・データベースへのパブリック・リンクまたはプライベート・リンクを作成します。データベース・リンクは、リモート・データベースのオブジェクトをローカル・データベースと同様にアクセスできるようにします。パブリック・リンクは、DBAとSYSADMだけが作成することができます。プライベート・リンクはCONNECT以上のセキュリティ権限をもつユーザーが作成することができます。

データベース・リンクは、リモート・データベースへの接続を作成し、リモート・データをローカル・データベースと同様にアクセスできるようにします。リンクには、リモート・データベースの識別情報だけでなくセキ

ュリティ情報をもたせることもできます。別のユーザー名やパブリック・リンク用のアカウントでリモート・データベースに接続することが許されます。

データベース・リンクを作成するには、リンク名とリモート・データベース名を指定します。ローカルおよびリモートの**dmconfig.ini**ファイルには、相手側のデータベース・セクションが含まれていなければなりません。各々のデータベース・セクションには、相手側のデータベース・サーバーのIPアドレスとポート番号が含まれていなければなりません。IPアドレスとポート番号は、DB_SVADRとDB_PTNUMキーワードで設定します。

PUBLIC／PRIVATEキーワードはオプションです。このキーワードは、パブリックまたはプライベートのデータベース・リンクのタイプを指定します。パブリック・リンクはデータベースの全ユーザーが使用することができ、プライベート・リンクはリンク作成者だけが使用することができます。DBAとSYSADMだけがパブリック・リンクを作成することができます。一方、プライベート・リンクは全てのユーザーが作成することができます。同名のパブリック・リンクとプライベート・リンクがあるときは、プライベート・リンクの方が使用されます。初期設定ではプライベート・リンクを作成します。

IDENTIFIED BYキーワードはオプションです。このキーワードは、リモート・データベースに接続するときに使用するユーザー名とパスワードを指定します。ユーザー名は、リモート・データベースのCONNECT以上のセキュリティ権限をもつ既存ユーザーの名前を与えます。リモート・データベースにリンク接続して実行できるオペレーションは、このユーザーのセキュリティ権限とオブジェクト権限に依存します。ユーザー名を指定しないときは、ローカル・データベースの現在のユーザー名を使用してリモート・データベースに接続します。

リンク名の最大長は32字です。文字、数字、アンダースコア、記号 $ と # を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1**　リモート・データベース**FieldOffice**のパブリック・データベース・リンク**FieldLink**を作成します。リンクの作成者は、ローカル・データベースの**DBA**か**SYSADM**セキュリティ権限をもち、同じ名前のユーザーがリモート・データベースに存在しなければなりません。このリンクを使用すると、自動的にリンク作成者と同じユーザー名でリモート・データベースに接続し、リモート・データベースのユーザーがもつセキュリティ権限とオブジェクト権限が与えられます：

```
CREATE PUBLIC Database LINK FieldLink CONNECT TO FieldOffice;
```

➲ **使用例2**　リモート・データベース**FieldOffice**のパブリック・データベース・リンク**FieldLink**を作成します。リンクの作成者は、ローカル・データベースの**DBA**か**SYSADM**セキュリティ権限をもっていなければなりません。このリンクを使用すると、自動的にユーザー名**LinkUser**、パスワード**dil3ryx9**でリモート・データベースに接続し、リモート・データベースのユーザーがもつセキュリティ権限とオブジェクト権限が与えられます：

```
CREATE PUBLIC Database LINK FieldLink CONNECT TO FieldOffice
                              IDENTIFIED BY LinkUser dil3ryx9;
```

➲ **使用例3**　リモート・データベース**FieldOffice**のプライベート・データベース・リンク**FieldLink**を作成します。リンクの作成者は、ローカルとリモートのデータベースに同じユーザー名をもっていなければなりません。このリンクを使用すると、自動的にリンク作成者と同じユーザー名でリモート・データベースに接続し、リモート・データベースのユーザーに許されているセキュリティ権限とオブジェクト権限が与えられます。同名のパブリック・リンクがあってもプライベート・リンクの方が使用されます：

```
CREATE PRIVATE Database LINK FieldLink CONNECT TO FieldOffice;
```

➲ **使用例4**　リモート・データベース**FieldOffice**のプライベート・データベース・リンク**FieldLink**を作成します。このリンクを使用すると、自動的にユーザー名**Vivian**、パスワード**a23456**でリモート・データベースに接続し、リモート・データベースのユーザーがもつセキュリティ権限とオブジェクト権限が与えられます。ローカルとリモートのユーザー名が違うときに

は、この方法が役に立ちます。同名のパブリック・リンクがあってもプライベート・リンクの方が使用されます：

```
CREATE PRIVATE Database LINK FieldLink CONNECT TO FieldOffice
                              IDENTIFIED BY Vivian a23456;
```

関連SQL

CLOSE DATABASE LINK

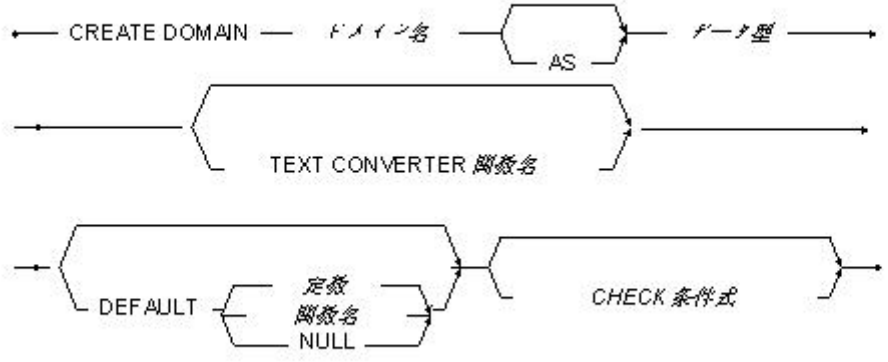
DROP DATABASE LINK

3.32 CREATE DOMAIN

目的

ドメインを新規に作成します。

構文



ドメイン名	作成するドメインの名前
データ型	ドメインに使用するデータ型
リテラル	カラム値を挿入しないときに用いるリテラル値
定数	カラム値を挿入しないときに用いる定数

関数名	カラム値を挿入しないときに用いる組み込み関数
制約名	設定する制約の名前
条件式	真または偽と評価される任意の式

説明

CREATE DOMAIN文は、ドメインを新規作成します。初期設定値とカラム制約のオプションがあります。RESOURCE以上のセキュリティ権限をもつユーザーなら誰でもCREATE DOMAINを実行することができます。

ドメインは、データ型、初期設定値、カラム制約をまとめたユーザー定義データ型です。ドメインは、CREATE TABLE文やALTER TABLE ADD COLUMN文のカラム定義でデータ型の代わりに使用し、カラムに挿入できる値の範囲を定義します。

例えば、DATEデータ型を基にして1900年1月1日から今日までの日付のみを受け入れ、初期設定値 NOW( )をもつドメインを作成することができます。ドメインを使用して定義されたカラムは、ドメインの特性を継承します。初期設定値やカラム制約をその都度指定せずに、一貫して同じデータ型のカラムを定義します。

ドメインを作成するには、データ型とオプションの初期設定値とカラム制約を指定します。SERIAL以外の任意のデータ型をドメインに使用することができます。初期設定値とカラム制約は、DEFAULTとCHECKキーワードを使用して指定します。

CREATE DOMAIN節のTEXT CONVERTER文でドメインを作成することができます。DBMasterはドメインのTEXT CONVERTER文が定義されているとき、テキスト索引とPURETEXT() UDF作成のためTEXT CONVERTER関数を使ってCLOB、NCLOB、BLOB、FILEデータをプレーンなテキストに変換します。TEXT CONVERTER関数名はBLOB関連型の引数を含む必要があります。戻り型はCLOB、BLOBデータ型で返されるかエラーが返答されます。最大で32767ドメインをTEXT CONVERTER文で作成できます。

DEFAULTキーワードはオプションです。このキーワードは、カラム値を与えずに行を挿入するときに、カラムに挿入される初期設定値を指定しま

す。初期設定値としては、リテラル値、定数、組み込み関数、NULLキーワードを使用することができます。ドメインの作成に使用する組み込み関数は、PI( )、NOW( )、USER( )のように引数の無いものだけにします。初期設定値としてNULLキーワードを使用しているドメインは、カラム定義にNOT NULLキーワードを付けることはできません。

CHECKキーワードはオプションです。このキーワードは、カラムに挿入できる値の範囲(制約)を指定します。受け入れ値の範囲を指定する条件式は、真または偽と評価される任意の式です。CHECKキーワードの条件式の中では、VALUEキーワードを使用して行のカラム値を表します。CHECK条件を満たさないときは、SQL文は処理されません。

ドメインで指定した初期設定値とカラム制約は、カラム定義で与えた初期設定値とカラム制約と同じ結果になります。ただし、カラム定義で与えた初期設定値はドメインの初期設定値を打ち消し、カラム定義で与えた制約はドメインの制約に追加されます。

カラム定義で与えたカラム制約がドメインのカラム制約と競合しないことを確かめておきます。ドメインでカラムを定義するときには、カラム制約が競合するかどうかはチェックされません。制約が競合すると、一部または全てのデータ挿入や更新が禁止されてしまうかもしれません。

ドメイン名の最大長は32字です。文字、数字、アンダースコア、記号 $ と # を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1　INTEGER**データ型を基に**AllNum**という名前のドメインを作成します：

```
CREATE DOMAIN AllNum AS INTEGER;
```

➲ **使用例2　INTEGER**データ型を基に初期設定値が**0**の**AllNum**という名前のドメインを作成します：

```
CREATE DOMAIN AllNum AS INTEGER DEFAULT 0;
```

➲ **使用例3　INTEGER**データ型を基に**NULL**値不可の**AllNum**という名前のドメインを作成します：

```
CREATE DOMAIN AllNum AS INTEGER CHECK VALUE IS NOT NULL;
```

➲ **使用例4　INTEGER**データ型を基に初期設定値が **0** で正の整数だけを入力することができる**PosNum**という名前のドメインを作成します：

```
CREATE DOMAIN PosNum AS INTEGER DEFAULT 0 CHECK VALUE >= 0 AND VALUE <= 100;
```

➲ **使用例5　DATE**データ型を基に初期設定値とカラム制約に**NOW( )**関数を使用する**ValidDate**という名前のドメインを作成します：

```
CREATE DOMAIN ValidDate AS DATE
                        DEFAULT NOW()
                        CHECK VALUE > '01/01/1900' AND VALUE <= NOW();
```

注：　ドメインを作成するときには引数を取らない関数だけが使用できます。

関連SQL

DROP DOMAIN

CREATE TABLE

ALTER TABLE ADD COLUMN

3.33 CREATE GROUP

目的

グループを新規に作成します。

構文

グループ名　　新規に作成するグループの名前

説明

CREATE GROUP文は、ユーザーのグループを新規に作成します。グループ内のユーザーは、グループがもつ全てのオブジェクト権限を獲得します。SYSADMかDBAだけがCREATE GROUPを実行することができます。

グループは、多数のユーザーをもつデータベースのオブジェクト権限の管理を容易にします。グループは、同じオブジェクト権限を必要とするユーザーをまとめるときに使用します。グループに与えられたオブジェクト権限は、自動的にグループ全員に与えられます。グループに含めるユーザーは、グループ作成後にADD TO GROUPを使用して追加します。

DBMasterでは、グループを更に別のグループのメンバーにする入れ子グループも使用できます。ただし、入れ子のグループが自分のグループに循環してはいけません。例えば、group2がgroup1のメンバーならば、group1をgroup2のメンバーにすることはできません。グループを別グループのメンバーに追加する方法は、ユーザーをグループに追加するのと同じです。

グループ名は、SYSTEM、PUBLIC、GROUP、既存のユーザー名やグループ名以外のものにします。グループ名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例**　**Manager**という名前のグループを新規に作成します：

```
CREATE GROUP Manager;
```

関連SQL

ADD TO GROUP

REMOVE FROM GROUP

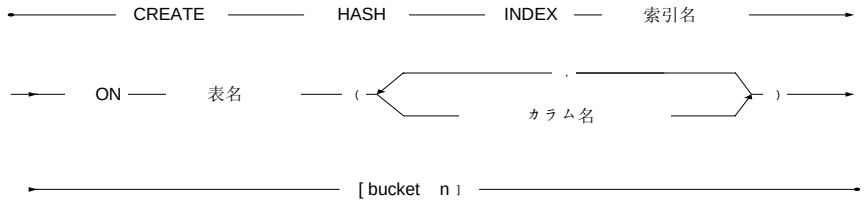
DROP GROUP

3.34 CREATE HASH INDEX

目的

テーブルにハッシュ索引を作成する。

構文



説明

ハッシュ索引は、メモリ表にのみ作成できます。 ハッシュ索引のメリットは、ハッシュ索引に保存されたデータにユーザーが素早くアクセスできることです。 ハッシュ索引は、イコール式とイコール結合パフォーマンスも向上させます。テーブルにハッシュ索引を作成するためにユーザーはCREATE HASH INDEX index_name ON table_name(column_name, …) [bucket n]を使用できます。ここで索引名は作成されているシャープ索引の名前で、表名はメモリ表の名前、カラム名は影響を受けているメモリ表のカラムの名前(この値はasc/descカラムを指定できません)で、バケットnは作成されているハッシュ表のアレイサイズを設定します。

索引名　　*作成する新規ハッシュ索引の名前*

表名　　*索引を作成しているメモリ表の名前*

カラム名　　*ハッシュ索引に作成されるカラムの名前*

バケットnはアレイサイズを設定します

使用例

➲ **使用例**　メモリ表を作成しているとき、ハッシュ索引**idx1**は**31**のアレイサイズを有するカラム**c01_int** と**c02_char**を使用して、メモリ表**tb_mem**に作成できます。

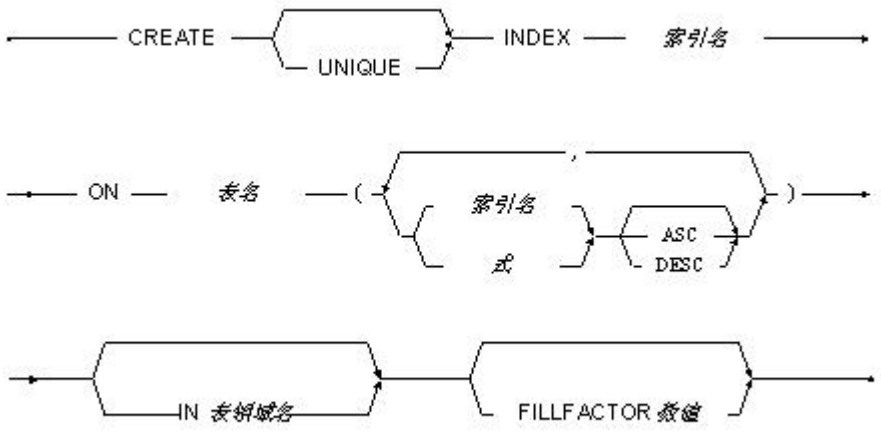
```
create hash index idx1 on tb_mem (c01_int, c02_char) bucket 31;
```

3.35 CREATE INDEX

目的

表の索引を新規に作成します。

構文



索引名	作成する索引の名前
表名	索引を作成する表の名前
カラム名	索引を作成するカラムの名前 (複数)
式	索引に作成された式
表領域名	索引を作成する表領域の名前
数値	フィルファクタの値

説明

CREATE INDEX文は、既存の表の索引を新規に作成します。索引は、表全体を検索せずに素早く特定の行の位置を決め、問合せのパフォーマンスを良くするのに使用します。表の所有者、DBA、INDEX権限をもつユーザーだけがCREATE INDEXを実行することができます。

索引は、キーと呼ばれるカラム(複数)の値から素早く特定の表行をアクセスできるようにするメカニズムです。索引は、表のキーカラムと同じデータをソートし構造化してもっており、表よりも高速に検索できるようにな

っています。索引を作成しても、その操作はユーザーには見えません。DBMSは、可能な限り索引を使用して問合せパフォーマンスを良くします。

索引を作成するには、索引名、索引を作成する表の名前、表のキーカラムの名前を指定します。最大32カラムまでの索引を作成することができます。表のどのようなカラムでも索引として使用できます。.索引の最大レコードサイズは4000バイトに制限されています。

索引作成によって頻繁に使われる式のパフォーマンス向上を図れます。XMLカラムの場合、XML UDF:extract() 、extractvalue()に索引を作成してXPathクエリの速度を上げます。extract() とextractvalue()の違いを理解する必要があります。asc/descや一意索引が使用できない場合でもextract()は複数、単一、ゼロの値を扱えます。extractvalue()はUDFの結果に単一またはゼロの値のみ扱えます。値が複数のとき、extractvalue()asc/descや一意索引が使用できる場合でも既存タプルへの索引作成に失敗し、タプルに新たにINSERTを行おうとすると失敗します。有効な索引構築には以下のXPathルールがあります;

- 述語を含まない
- 関数を含まない
- 完全なロケーションパスが含まれる
- ‘child’ axisのみ許可
- 結果ノードセットにはリーフノードのみが含まれる(Simple Type要素ノードもしくは属性ノード)
- qnameはすべて要素ノードに一致しなければなりません
- 各属性の名前はすべて属性ノードと一致しなければなりません
- 属性ノードもしくは要素リーフノードに基づく必要があります
- '/order/items/item/@product' や'/order/date'といった複合ノンリーフノード、コメントノードにはできません。
- 許可されない位置'/order/items/item[1]/@product'

- 使用可能関数'count(/order/items/item)'
- 式は使用不可

UNIQUEキーワードはオプションです。このキーワードは、索引が一意かどうかを指定します。一意索引は、二つ以上の行が同じキー値をもつことを禁止します。ただし、NULLは一意値とみなされ、一意索引の中にNULLをもつ複数の行があってもかまいません。表が空でないときに一意索引を作成すると、全てのキーが異なっているかチェックされます。重複キーがあればエラーメッセージが返され、索引は作成されません。一意索引をもつ表にレコードを挿入/更新すると、挿入/更新レコードと同じキー値をもつ既存レコードが無いかがチェックされます。初期設定では一意索引を作成しません。一意索引を作成するときは、UNIQUEキーワードを指定します。

ASC／DESCキーワードはオプションです。これらのキーワードは、索引のソート順が昇順か降順かを指定します。ソート順はカラム毎に指定します。ある索引カラムを昇順にする一方で、他を降順にすることができます。索引のソート順が問合せの出力順序に影響する場合があります。索引が降順ならば、問合せで何も指定しなくても、出力が降順になる可能性があります。特定のソート順が必要な問合せには、ORDER BY句を使用して出力のソート順を指定します。初期設定のソート順はASCです。

FILLFACTORキーワードはオプションです。このキーワードは索引ページの使用率を指定します。フィルファクタは、既存レコードの更新用にスペースを確保しておき、索引ページの使用を最適化するようにします。1%～100%の範囲内にフィルファクタを指定します。索引作成後に表を頻繁に更新する場合は、表を索引化後、低いフィルファクタ (例えば 50)を指定し、新しいキーの挿入スペースを残しておきます。表を更新することがほとんど無ければ、フィルファクタを初期設定値の100にします。

表にデータをロードすると、レコードが挿入される毎に表の全ての索引が更新されます。このため、索引を作成する前に、全てのデータをロードするようにします。データをロードした後に索引を作成した方が、ロード前に索引を作成するよりもはるかに効率的です。

マスターの表と異なる表領域に索引を作成することもできます。表領域を指定しない場合、マスター表と同じ表領域に索引が作成されます。

索引名は表毎に一意にします。索引名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1　Employeesinfo**表の**FName、LName**カラムに索引**NameIndex**を作成します。索引は一意ではなく重複値を含みます：

```
CREATE INDEX NameIndex ON Employeesinfo (FName, LName);
```

➲ **使用例2　Employeesinfo**表の**FName、LName**カラムに索引**NameIndex**を作成します。両カラムとも降順にソートされます：

```
CREATE INDEX NameIndex ON Employeesinfo (FName DESC, LName DESC);
```

➲ **使用例3　Classes**表の**Course、Section**カラムに一意索引**ClassIndex**を作成します。索引には重複値が許されません：

```
CREATE UNIQUE INDEX ClassIndex ON Classes (Course, Section);
```

➲ **使用例4　Classes**表の**Course、Section**カラムに一意索引**ClassIndex**を作成します。索引のフィルファクタは**80**です。索引には重複値が許されません：

```
CREATE UNIQUE INDEX ClassIndex ON Classes (Course, Section) FILLFACTOR 80;
```

➲ **使用例5**　以下は**Classes**テーブルの**concat(Course, Section)** カラムで**ExprIndex** という名前の一意インデックスを作成します; インデックスは複製値を含まず、フィルファクタを**80**に設定します：

```
CREATE UNIQUE INDEX ExprIndex ON Classes (concat(Course,Section)) FILLFACTOR
80
```

関連SQL

ALTER TABLE PRIMARY KEY

ALTER TABLE FOREIGN KEY

CREATE TABLE

CREATE TABLESPACE

CREATE TEXT INDEX

DROP INDEX

REBUILD INDEX

3.36 CREATE PROCEDURE

CREATE OR REPLACE PROCEDURE文は新しいストアドプロシージャを作成して既存なプロシージャを代わりるため使用されます。ストアドプロシージャを通じてデータベースエンジンは重複にコンパイル、SQLコマンドを最適化にする必要がありません。頻繁に重複のタスクを減少して性能を向上されます。RESOURCE 権限以上のユーザー、SQL構文を実行する安全とオブジェクト権限を持つユーザーはCREATE OR REPLACE PROCEDURE文が使用できます。

ストアド・プロシージャは、コンパイルされた実行形式のSQLデータ操作文で永続的にデータベースに格納されます。ストアド・プロシージャは、コンパイル・最適化せずにSQLデータ操作文を繰り返し実行できるようにします。ストアド・プロシージャはストアド・コマンドと似ていますが、一つの SQLデータ操作文だけからなり、プログラムロジックをもたない点が異なります。

ストアド・プロシージャを作成するには、プロシージャ名と有効なSQLデータ操作文SELECT、INSERT、UPDATE、DELETEを指定します。データ操作文のカラム値の位置にホスト変数を使用して、プロシージャ実行時に実際のカラム値を割当てることができます。ホスト変数を使用するには、データあるいはカラムの値を疑問符(?)で置き換えます。

FROM FILE

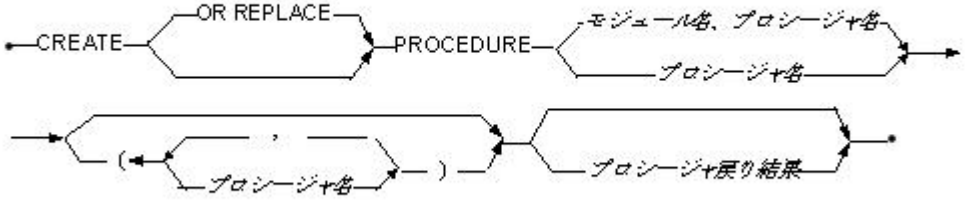
OR REPLACE　OR REPLACEで既存なプロシージャを再び作成します。この句を使用して既存なプロシージャの定義を変更できます

ファイル名　作成するプロシージャのファイル名

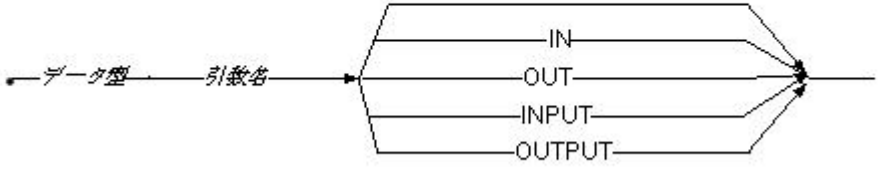
➲ **使用例**　プロシージャを作成、再作成します

```
dmSQL> CREATE PROCEDURE FROM ‘file-name’;
dmSQL> CREATE OR REPLACE PROCEDURE FROM ‘file-name’;
```

ESQL SP



<プロシージャ引数名>句



OR REPLACE	OR REPLACEで既存なプロシージャを再び作成します。この句を使用して既存なプロシージャの定義を変更できます
モジュール名	作成するプロシージャのモジュール名
プロシージャ名	作成するプロシージャ名
プロシージャ引数	作成するプロシージャの引数

プロシージャ戻り結果　作成するプロシージャからの結果セット
注：プロシージャにCREATE OR REPLACE COMMANDとCREATE OR REPLACE PROCEDUREコマンドをサポートしません。
注：AUTOCOMMITはOFFに設定すると、CREATE OR REPLACE PROCEDUREコマンドをサポートしません。

➲ **使用例**　プロシージャを作成するまたは代わります.

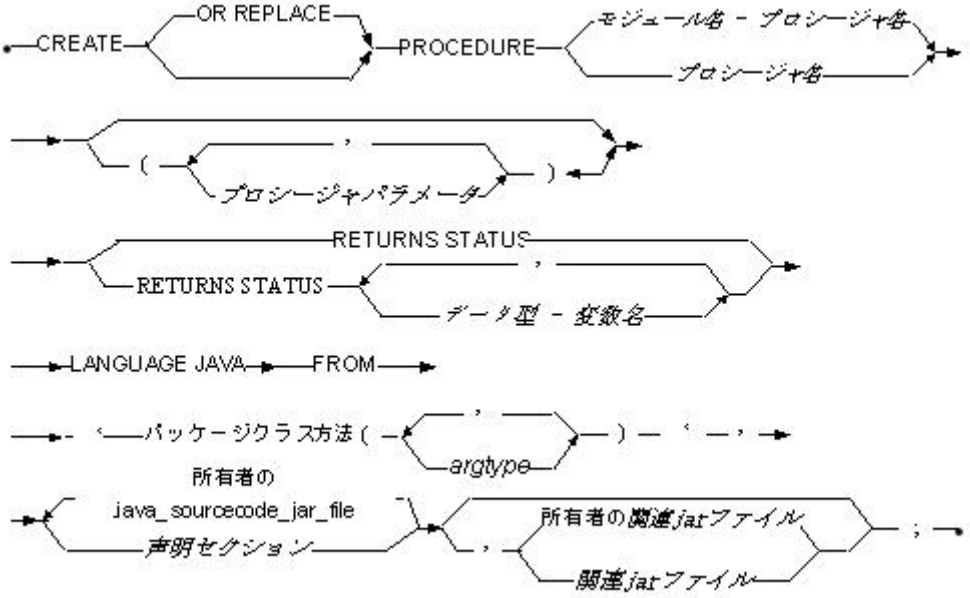
- ファイルから ESQL SPを作成する:

```
dmSQL> CREATE OR REPLACE PROCEDURE FROM ‘proc1.ec’;
```

- ec ファイルを書きます:

```
EXEC SQL CREATE OR REPLACE PROCEDURE proc1 (char(10) i1, char(10) i2 output)
 returns char(10) o1, char(10) o2;
{
        EXEC SQL BEGIN CODE SECTION;
        EXEC SQL select FName from tb_staff where LName =:i1 into:i2;
        EXEC SQL returns select * from tb_staff into :o1,:o2;
        EXEC SQL END CODE SECTION;
}
```

JAVA SP



OR REPLACE	OR REPLACEで既存なプロシージャを再び作成します。この句を使用して既存なプロシージャの定義を変更できます
モジュール名	作成するプロシージャのモジュール名
プロシージャ名	作成するプロシージャ名
プロシージャ引数	作成するプロシージャの引数
データ型	戻り変数のデータ型
変数名	戻り変数の名
パッケージクラス方法	作成するプロシージャのjava方法
argtype	java方法の引数型
java_sourcecode_jar_ile	javaソースコードの物理jarファイル
関連jarファイル	ロジックjarファイル

➲ **使用例**　プロシージャを作成するまたは代わります

- java ファイルを書きてAddStaff.javaに追加します

```
package staff;
import java.sql.*;

public class AddStaff
{
        // Add an row into the tb_staff table
        public static void addStaff(String fName, String lName)
        throws Exception
        {
                        // Register DBMaker JDBC Driver
                        Class.forName("dbmaker.sql.JdbcOdbcDriver");
                        // Connect to database
                        Connection                  conn                  =
DriverManager.getConnection("jdbc:default:connection");

                        // Prepare SQL statement
                        PreparedStatement              pstmt              =
conn.prepareStatement("insert into tb_staff values(?,?)");

                        // Set values of the dynamic SQL argument
                        pstmt.setString(1, fName);
                        pstmt.setString(2, lName);

                        // Execute the dynamic SQL statement
                        pstmt.execute();

                        // Close the dynamic SQL statement
                        pstmt.close();
```

```
                        // Close the connection
                        conn.close();
        }
}
```

DOS コマンドラインにこのAddStaff.javaファイルをコンパイルして、既存なディレクトリにAddStaff.classファイルを作成されます。

```
javac AddStaff.java
```

AddStaff.classをdir\staffまでコピーします。

このクラスをパックします、既存なディレクトリにStaff.jarファイルを作成されます。

```
jar cvf addStaff.jar   staff\AddStaff.class
```

<DB_SPDIR>を使用してjar\SYSADMディレクトリを作成します。ファイルaddStaff.jarを<DB_SPDIR>\jar\SYSADMまで移動します。

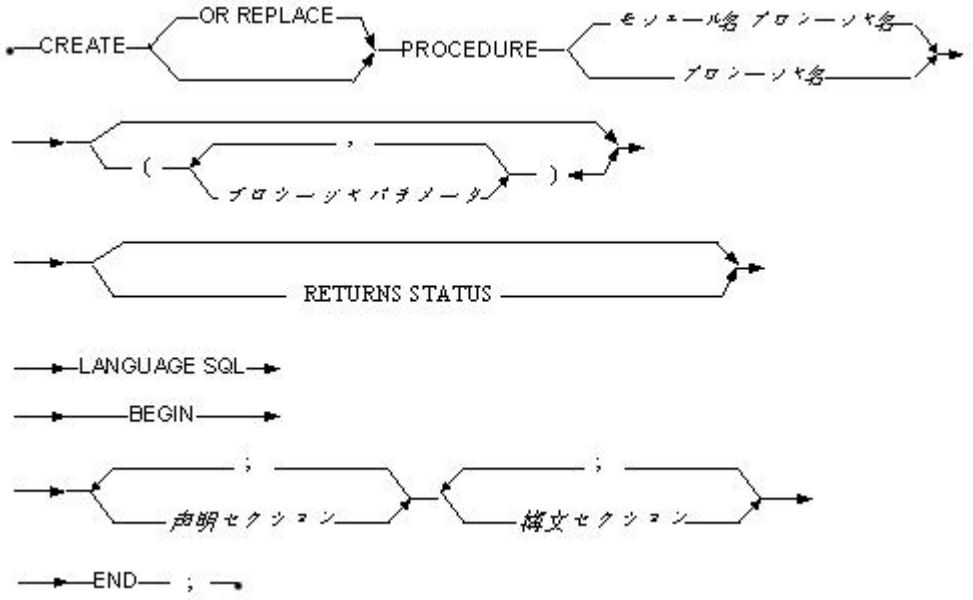
jarfileを追加します：

```
dmSQL> ADD JARFILE addStaff addStaff.jar;
```

Java SPを作成して、再作成します：

```
dmSQL> CREATE OR REPLACE PROCEDURE addStaff(char(12) fname,char(12) lname)
RETURNS STATUS LANGUAGE JAVA FROM
 'staff.AddStaff.addStaff(String,String)',addStaff;
```

SQL SP



OR REPLACE	OR REPLACEで既存なプロシージャを再び作成します。この句を使用して既存なプロシージャの定義を変更できます
モジュール名	作成するプロシージャのモジュール名
プロシージャ名	作成するプロシージャ名
プロシージャ引数	作成するプロシージャの引数
データ型	戻り変数のデータ型
変数名	戻り変数の名
声明セクション	作成するプロシージャの主な声明句のセクション
構文セクション	作成するプロシージャの主な句

*注：　プロシージャに**CREATE OR REPLACE COMMAND**と**CREATE OR REPLACE PROCEDURE**コマンドをサポートしません。*

注：　AUTOCOMMIT はOFF に設定すると、CREATE OR REPLACE PROCEDURE コマンドをサポートしません。

➲ **使用例**　プロシージャを作成するまたは代わります:

- ファイルからSQL SP を作成します:

```
dmSQL> CREATE OR REPLACE PROCEDURE FROM ‘proc1.sp’;
```

- spファイルを書きます:

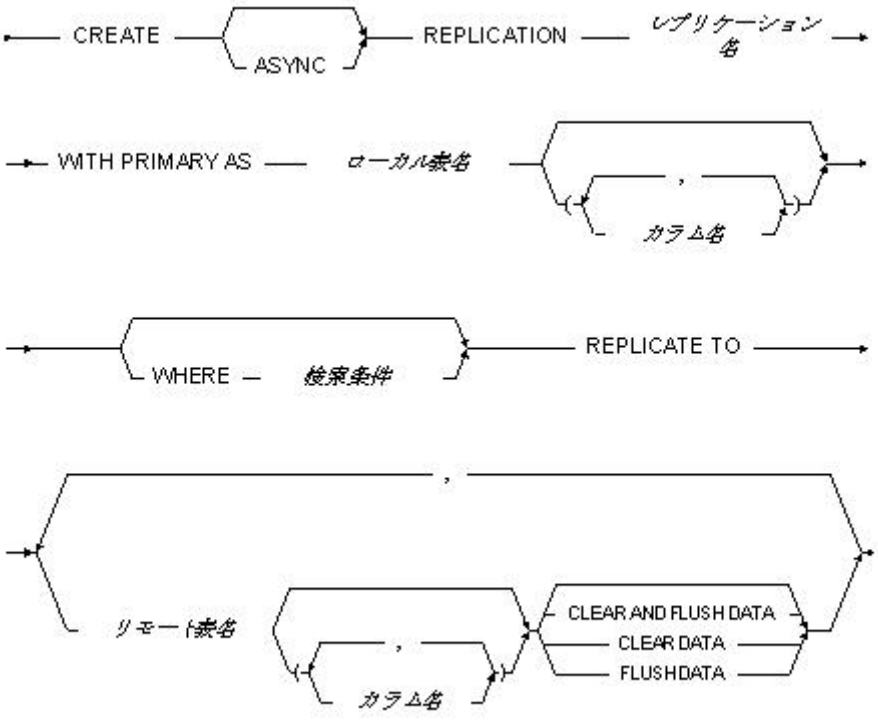
```
CREATE OR REPLACE PROCEDURE proc1
LANGUAGE SQL
BEGIN
     DECLARE cur CURSOR WITH RETURN FOR select * from tb_staff;
     OPEN cur;
END;
```

3.37 CREATE REPLICATION

目的

表に表レプリケーションを新規に作成します。

構文



レプリケーション名	作成する表レプリケーションの名前
ソース表名	レプリケートするソース表の名前
カラム名	1. ソース表のカラムの名前
	2. ターゲット表のカラムの名前
検索条件	レプケートされる行が満たすべき条件
リモート表名	リモート・データベースの表の名前

説明

CREATE REPLICATION文は、表に新規レプリケーションを作成します。一時表には、レプリケーション、シノニム、ビューを作成することはできません。表の所有者、DBA、SYSADMだけがCREATE REPLICATIONを実行することができます。

レプリケーションは、表全体または一部のカラムを遠隔地にコピーします。遠隔地のユーザーは、レプリケートされたデータで作業できるようになります。レプリケートされたコピーは、他の場所のデータベースと同期が取られます。各データベースは、低速ネットワーク接続を経由して別のマシンに行かずに、直接効率的にデータ要求に応じることができるようになります。レプリケーションは、ターゲット側にデータベースのバックアップを取るのとは異なり、ユーザーの介入無しにDBMS自身によってトランザクション毎に同期が取られます。

レプリケーションには同期と非同期があります。CREATE REPLICATIONは、同期・非同期レプリケーションを作成します。同期レプリケーションは、ソース表の変更と同時にターゲット表を変更します。非同期レプリケーションは、ソース表の変更を格納しておき、予め定義されたスケジュールに従ってターゲット表を変更します。

DBMasterの同期レプリケーションは、グローバルトランザクションモデルを使用します。このモデルでは、ターゲット表へのデータ・レプリケーションがソース・トランザクションの不可欠な部分として取り扱われます。これは、ターゲット・データベースへのデータ・レプリケーションに失敗すると、ソース表のトランザクションも失敗することを意味します。

DBMasterの非同期レプリケーションは、トランザクションログを使用してターゲット表にデータをレプリケートします。ソース表の変更はトランザクションログに格納され、予め決められたスケジュールに従ってターゲット表にレプリケートされます。ソース・トランザクションとターゲット・トランザクションは、トランザクションログを使用することによって独立に取り扱われ、ターゲットに接続できなくてもソース表を更新することができるようになります。非同期レプリケーションは、ネットワークやター

ゲット・データベースの障害に耐えることができます。レプリケーションは、障害が復旧するまで再試行されます。

レプリケーションを作成するには、レプリケーション名、ソース表名、レプリケーと先のターゲット表名を指定します。ソース表とターゲット表は、各々のデータベースに存在するものでなければなりません。表が削除されると、表に作成されているレプリケーションも自動的に削除されます。

ソース表のカラムリストを指定しないと、表全体がレプリケートされます。表全体をレプリケートするときは、ソースとターゲットの表のカラムは同じ名前とデータ型をもっていなければなりません。ソース表のカラムリストのカラムは、左から右に対応するターゲット表のカラムリストのカラムにレプリケートされます。ソースとターゲットの両方のカラムリストを与えて、ソース表とターゲット表の対応カラムを明示的に指定します。いずれの場合も、主キーカラムをレプリケーションに加えます。両方の表の主キーカラムのカラム数とデータ型は、一致していなければなりません。

DBMasterは、レプリケーションを表に関連付けます。所有者名とオブジェクト名を組み合わせた完全修飾名でレプリケーションを識別することはしません。同じ表上の全てのレプリケーション名は一意でなければなりません。同期レプリケーションは、ソース表の所有者のセキュリティ権限とオブジェクト権限で動作します。ただし、ターゲット表がリンクで指定されている場合は、レプリケーションはリンクと同じセキュリティ権限とオブジェクト権限で動作します。非同期レプリケーションは、CREATE SCHEDULE文のIDENTIFIED BY句で指定したターゲット・データベースのユーザーのセキュリティ権限とオブジェクト権限で動作します。

ASYNCキーワードはオプションです。このキーワードは、非同期レプリケーションを作成する指定です。非同期レプリケーションを作成する前に、ターゲット表があるターゲット・データベースにレプリケーション・スケジュールを作成しておきます。このキーワードを使用しないと、同期レプリケーションが初期設定で作成されます。

WHEREキーワードはオプションです。このキーワードは、データをターゲット表にレプリケートするときの検索条件を指定します。検索条件を満

たす行だけがレプリケートされます。詳細については、SELECT文のWHERE句の説明を参照してください。

CLEAR DATA／FLUSH DATA／CLEAR AND FLUSH DATAキーワードはオプションです。これらのキーワードはレプリケーション作成時に実行するオペレーションを指定します。CLEAR DATAキーワードは、ターゲット表の全データを削除します。FLUSH DATAキーワードは、検索条件にマッチするデータをターゲット表にコピーします。CLEAR AND FLUSH DATAキーワードは、ターゲット表の全データを削除してから検索条件にマッチするデータをターゲット表にコピーします。

NO CASCADEキーワードは、非同期レプリケーションにのみ有効なオプションです。このキーワードは、カスケードレプリケーションかどうかを指定します。例を使用して、カスケードレプリケーションを説明します。大多数の組織の指令は、最上位から基本レベルに流れます。データをAからBに、次にCにレプリケートするのもこれと同じで、典型的なカスケードレプリケーションの一種です。非カスケードモデルでは、AはBにデータをレプリケートし、BはAにデータをレプリケートします。このようなデータモデルならば、NO CASCADE オプションをONにすることができます。初期設定の指定はCASCADEです。

非同期レプリケーションは、非同期レプリケーションが参照する表やカラムを削除したり、表のカラム定義を変更したり、BEFORE、AFTERキーワードを使用して表にカラムを追加すると不正になり、二度と使用することができなくなります。BEFORE、AFTERキーワードを使用せずに表にカラムを追加した場合は、非同期レプリケーションには影響しません。非同期表レプリケーションは、表を変更しても影響されません。不正なレプリケーションは、データベースから削除して削除します。表を削除すると、表に作成されたレプリケーションも自動的に削除されます。

レプリケーション名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1**　ソース表**Employeesinfo**のレプリケーション**EmpRep**を作成します。ターゲット・データベース**FieldOffice** は、ソース**dmconfig.ini**ファイルのデータベース構成セクションで識別できるようにしておきます。ターゲット表名も**Employeesinfoです。両方の表の全てのカラム名とデータ型は同じです：**

```
CREATE REPLICATION EmpRep WITH PRIMARY AS Employeesinfo
                          REPLICATE TO FieldOffice:Employeesinfo;
```

➲ **使用例2**　上の例と同様ですが、ターゲット表のデータを全て削除してから、ソース表のデータをターゲット表にレプリケートします：

```
CREATE REPLICATION EmpRep WITH PRIMARY AS Employeesinfo
                          REPLICATE TO FieldOffice:Employeesinfo
                          CLEAR AND FLUSH DATA;
```

関連SQL

ALTER REPLICATION ADD REPLICATE

ALTER REPLICATION DROP REPLICATE

ALTER SCHEDULE

CREATE DATABASE LINK

CREATE SCHEDULE

DROP REPLICATION

DROP SCHEDULE

SUSPEND SCHEDULE

RESUME SCHEDULE

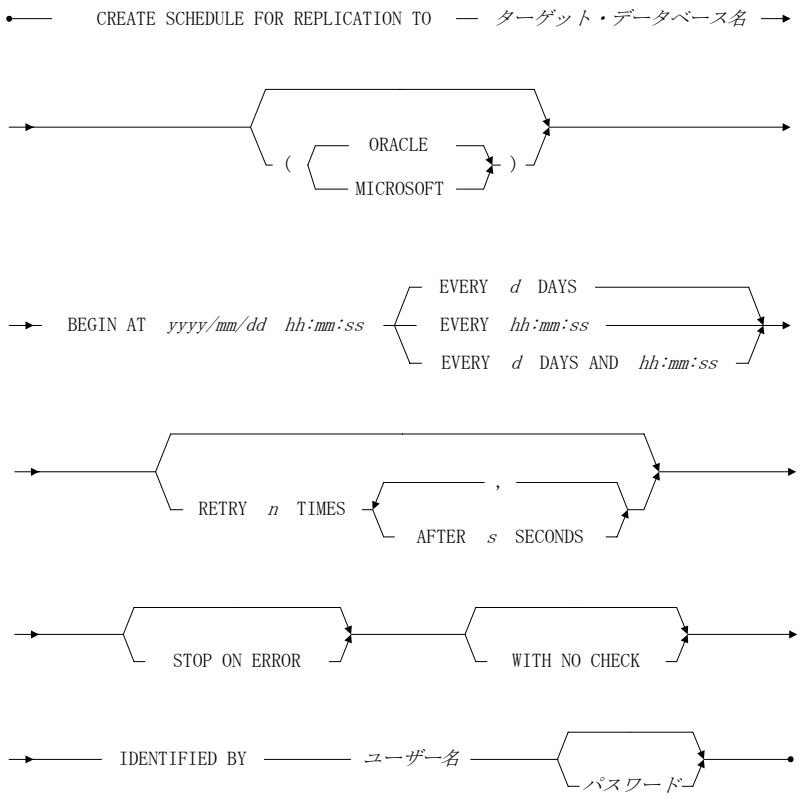
SYNCHRONIZE SCHEDULE

3.38 CREATE SCHEDULE

目的

非同期レプリケーションのスケジュールを作成します。

構文



ターゲット・データベース名　レプリケーション・スケジュールを作成するターゲット・データベースの名前 (リンク名は不可)

yyyy/mm/dd	レプリケーションの開始日
hh:mm:ss	1. レプリケーションの開始時刻 2. レプリケーションの間隔時間
d	ターゲット表のレプリケーション間隔日数
n	失敗したときの再試行回数
s	失敗したときの再試行間隔の秒数
ユーザー名	ターゲット・データベースのユーザー名
パスワード	ターゲット・データベースユーザーのパスワード

説明

CREATE SCHEDULE文は、非同期レプリケーションのスケジュールを作成します。同期レプリケーションはスケジュールを使用しないので、CREATE SCHEDULEは、同期レプリケーションには何も影響を与えません。DBAまたはSYSADMだけがCREATE SCHEDULEを実行することができます。

レプリケーションは、表全体または一部のカラムを遠隔地にコピーします。遠隔地のユーザーは、コピーされたデータで作業できるようになります。レプリケートされたコピーは、他の場所のデータベースと同期が取られます。各データベースは、低速ネットワーク接続を経由して別のマシンに行かずに、直接効率的にデータ要求に応じることができるようになります。レプリケーションは、ターゲット側にデータベースのバックアップを取るのとは異なり、ユーザーの介入無しにDBMS自身によってトランザクション毎に同期が取られます。

NO CASCADEキーワードは、非同期レプリケーションにのみ有効なオプションです。このキーワードは、カスケードレプリケーションかどうかを指定します。例を使用して、カスケードレプリケーションを説明します。大多数の組織の指令は、最上位から基本レベルに流れます。データをAからBに、次にCにレプリケートするのもこれと同じで、典型的なカスケードレプリケーションの一種です。非カスケードモデルでは、AはBにデータをレ

プリケートし、BはAにデータをレプリケートします。このようなデータモデルならば、NO CASCADEオプションをONにすることができます。初期設定の指定はCASCADEです。

DBMasterは、Oracle、Microsoft SQL Server等のデータベースと非同期レプリケーションを行うことができます。この種のレプリケーションは、DBMasterが異種環境の他のデータベースと共存することを可能にし、異種表レプリケーションと呼ばれます。異種環境のスケジュールを作成するときは、ORACLE、MICROSOFTキーワードでレプリケートするDBMSのタイプを指定します。レプリケーション・データは、第三者ターゲット・データベースに送信する前に前処理されます。ORACLEはターゲット・データベースOracle、MICROSOFTターゲット・データベースMicrosoft SQL Server を表します。

異種表レプリケーションを作成するときは、CLEAR DATA、FLUSH DATA、CLEAR AND FLUSH DATAキーワードを使用することはできません。レプリケーションを開始する前に、手動で第三者ターゲット・データベースのデータを削除・挿入して表を初期化しておきます。また、第三者ターゲット・データベースのスキーマチェックは行われません。スキーマをチェックし、ターゲット表とソース表のカラムとデータ型の互換性を確かめておきます。異種表レプリケーションのスケジュールを作成するときは、WITH NO CHECKキーワードを使用し、DBMasterがスキーマチェックを実行しないようにします。（この節の後にあるWITH NO CHECKキーワードの説明を参照。）DBMasterは、ODBC Driver Managerを使用して異種表レプリケーションを実行します。DBMasterサーバーはWindows NTでなければなりません。第三者リモートデータベースは、WindowsまたはUNIXプラットフォームのどちらでもかまいません。

BEGIN ATキーワードは、非同期レプリケーションの開始日時を指定します。日付フォーマットはyyyy/mm/ddで、yyyyは1970～2038の年、mmは01～12の月、ddは01～31の日です。時刻フォーマットは *hh:mm:ss*で、*hh* は00～23の時、*mm* は00～59の分、*ss* は00～59の秒です。年は1970～2038の範囲になければなりません。BEGIN ATキーワードには日付と時刻を含めます。既にレプリケーションが開始されているときに開始日時を将来の日付に変更すると、ターゲット・データベースにレプリケートされていない表データは、新しいレプリケーションの開始日時まで待たされます。

EVERYキーワードは、非同期レプリケーションの間隔を定義します。レプリケーション間隔は、時・分・秒、日数、または両方の組み合わせで与えることができます。EVERY *hh:mm:s*は時・分・秒毎のレプリケーションを指定します。EVERY *d* DAYSは日数毎のレプリケーションを指定します。*d*は1～365の範囲の日数です。EVERY *d* DAYS AND *hh:mm:ss* は両方の組み合わせを指定します。

RETRYキーワードは、SQL文の実行エラー、ロック・タイムアウト、ジャーナルフルに起因するセーブポイントへのロールバック等が発生したときに、表データ・レプリケーションの再試行回数を指示します。RETRY *n* TIMES は、0～2147483647の範囲の再試行回数を指定します。初期設定値は0です。RETRYキーワードを使用しないと、SQL文の処理中にネットワークエラー、リモートデータベースエラー、トランザクションをロールバックしなければならないエラー等によってレプリケートできなかった表データは、次のレプリケーション・スケジュールまで送信されません。

AFTERキーワードはオプションです。このキーワードはRETRYキーワードと一緒に使用し、エラー発生時の再試行間隔を指定します。AFTER *s* SECONDSは、0～2147483647の範囲の秒数で再試行間隔を指定します。初期設定値は5です。

STOP ON ERRORキーワードはオプションです。これらのキーワードは、ターゲット・データベースにレプリケートできないようなデータ更新が行われた場合のアクションを指定します。例えば、既にターゲット表から削除されているレコードを削除しようとした、或いは既に存在するレコードをターゲット表に挿入しようとした場合です。この種のエラーが発生したときに、STOP ON ERRORはデータのレプリケーションを停止します。このオプションを指定しない場合、エラーデータを無視して残りのデータのレプリケーションを続けることを指示します。初期設定では、このオプションを指定しません。

WITH NO CHECKキーワードはオプションです。現在は第三者データベースのスキーマをチェックすることができないので、異種表レプリケーションを作成するときはこのキーワードを使用します。WITH NO CHECKキーワードを使用する場合は、ユーザーがスキーマチェックの責任をもちます。ユーザーは、ターゲット表とソース表のカラムとデータ型が互換性を

もつことを確かめなければなりません。DBMasterからDBMasterへの同種表レプリケーションでは、WITH NO CHECKキーワードは必要ありません。

IDENTIFIED BYキーワードは、ターゲット・データベースに接続するときに使用するユーザー名とパスワードを指定します。指定されたターゲット・データベースのユーザーは、ターゲット表にINSERT、DELETE、UPDATE する権限をもたなければなりません。ターゲット・データベースに表データをレプリケートするときに実行できるオペレーションは、このユーザーのセキュリティ権限とオブジェクト権限に依存して定まります。

使用例

- **使用例1**　ターゲット・データベース**EmpRep** のレプリケーション・スケジュールを作成します。レプリケーションの開始日時は将来の日付と時刻を設定し、間隔は7日と**12**時間です。この時点でレプリケートされていない表データは、新しい開始日時まで待ってレプリケートされます：

```
CREATE SCHEDULE FOR REPLICATION TO EmpRep
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00;
```

- **使用例2**　使用例1と同じスケジュールを作成し、エラー、ロック・タイムアウト、フルジャーナルに起因するセーブポイントへのロールバックが発生したときの再試行回数3回と再試行間隔5秒も設定します：

```
CREATE SCHEDULE FOR REPLICATION TO EmpRep
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS;
```

- **使用例3**　使用例2と同じスケジュールを作成し、ターゲット・データベースのデータがレプリケートできないように更新されたときのアクションを**STOP**にします：

```
CREATE SCHEDULE FOR REPLICATION TO EmpRep
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS
                STOP ON ERROR;
```

➲ 使用例4　使用例3と同じスケジュールを作成し、ターゲット・データベースに接続するときのユーザー名とパスワードを**RepUser**と**rdejpe88**に設定します：

```
CREATE SCHEDULE FOR REPLICATION TO EmpRep
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS
                STOP ON ERROR
                IDENTIFIED BY RepUser rdejpe88;
```

➲ 使用例5　異種表レプリケーションです。**WITH NO CHECK**キーワードを指定してターゲット・データベースのスキーマチェックをしないようにします。ユーザーは、ターゲット表とソース表のカラムとデータ型が互換性をもつことを確かめまておきます。使用例4と同じスケジュールを作成し、**ORACLE**キーワードを使用してターゲット表が*Oracle 8.0*データベースにあることを指示します：

```
CREATE SCHEDULE FOR REPLICATION TO EmpRep (ORACLE)
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS
                STOP ON ERROR
                WITH NO CHECK
                IDENTIFIED BY RepUser rdejpe88;
```

関連SQL

ALTER SCHEDULE

CREATE REPLICATION

DROP REPLICATION

DROP SCHEDULE

RESUME SCHEDULE

SYNCHRONIZE SCHEDULE

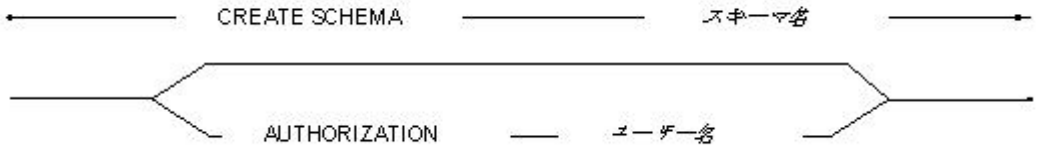
SUSPEND SCHEDULE

3.39 CREATE SCHEMA

目的

CREATE SCHEMAコマンドは新しいスキーマを現在のデータベース・システムに作成します。

構文



説明

CREATE SCHEMAコマンドは新しいスキーマを作成し、現在のデータベース・システムに入れます。 スキーマは基本的には名前領域です: スキーマ・オブジェクトとしても知られている命名されたオブジェクト(表、ビュー、索引、シノニム、トリガー、ドメイン、コマンド、プロシージャ)を含みますが、その名前は他のスキーマに存在するほかのオブジェクトの名

前に重複することがあります。 スキーマ・オブジェクトは、接頭辞としてスキーマ名を持つそれらのオブジェクトの名前を修飾することによってアクセスできます。

RESOUCE権以上を有するユーザーのみがスキーマを作成できます。 スキーマを作成中にuser_nameを省略すると、スキーマ製作者は所期設定のユーザーとなります。DBA権を有するユーザーのみが、自分自身以外のユーザーによって所有されるスキーマを作成できます。

ユーザーがDBMasterに対して接続権を与えられているとき、DBMasterはそのユーザーに対する所期設定のスキーマを作成します。 スキーマ名がユーザーの名前となります。 スキーマ名は一意でなければなりません。 同じ名前のスキーマがデータベースにすでに存在すると、エラーが返されます。

スキーマの所有者は次のように決定されます:

- AUTHORIZATION条件項が指定されている場合、指定されたユーザー名がスキーマ所有者です。 スキーマ名を省略する場合、指定されたユーザー名はスキーマ名として使用されます。
- AUTHORIZATION条件項が指定されていない場合、CREATE SCHEMA文を発行するユーザーがスキーマ所有者です。

使用例

➲ 使用例1

RESOURCE権を有するユーザーYUBINが、スキーマ**schm_def**を作成します。 YUBINはスキーマの所期設定所有者です。

```
CREATE SCHEMA schm_def;
```

➲ 使用例2

DBA権を有するユーザーが、所有者としてユーザーYUBINでスキーマを作成します。スキーマが作成されたときスキーマ名が指定されなかったため、YUBINが所期設定のスキーマ名になります。

```
CREATE SCHEMA AUTHORIZATION YUBIN
```

NOTE ユーザーが接続ステータスを与えられているとき、DBMasterはユーザー名となるスキーマ名を持つユーザーに対して、スキーマを自動的に作成することを覚えておくのは大切です。 同じ名前のスキーマがデータベースにすでに存在すると、エラー・メッセージが返されます。

➲ 使用例3

DBA権を有するユーザーが、所有者としてユーザーYUBINでスキーマ**schm_auth**を作成します。

```
CREATE SCHEMA schm_auth AUTHORIZATION YUBIN
```

➲ 使用例4

RESOURCE権を有するユーザーが、スキーマinventoryを作成します。 次に、スキーマに対してスキーマ・オブジェクトinventory.part と partindを作

成します。 作成された表のユーザーYUBINに完全なユーザー権を与えます。 ユーザーYUBINは、スキーマinventoryに対していかなる権限も持っていません。

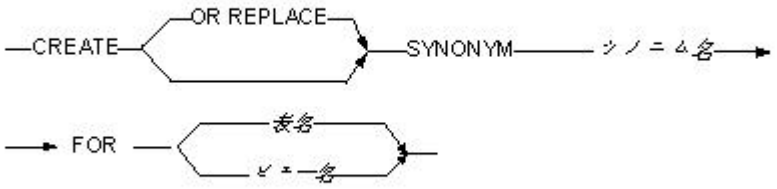
```
CREATE SCHEMA inventory;
CREATE TABLE inventory.part (partNo smallint not null, quantity int);
CREATE INDEX partind ON inventory.part (partNo);
GRANT ALL ON inventory.part TO YUBIN;
```

3.40 CREATE SYNONYM

目的

表またはビューのシノニムを新規に作成します。

構文



OR REPLACE	OR REPLACEで既存なトリガーを再び作成します。この句を使用して既存なトリガーの定義を変更できます
シノニム名	作成するシノニムの名前
表名	シノニムを作成する表の名前

説明

CREATE SYNONYM文は、既存の表またはビューのシノニムを作成します。一時表または別のシノニムにシノニムを作成することはできません。表またはビューの所有者、DBA、SYSADMだけがCREATE SYNONYMを実行する権限をもっています。

表やビューは、通常、所有者名とオブジェクト名の複合である完全修飾名によって識別されます。シノニムは、完全修飾名の表やビューの使用を簡明にする助けをします。

シノニムは、表名やビュー名の代わりに使用することができる別名です。システムカタログにシノニム定義を格納するストレージだけを必要とします。シノニムを使用することによって、完全修飾名を使用せずに表やビューをアクセスすることができます。

シノニム名が重複しない限り、表やビューに複数のシノニムを作成することができます。ユーザーは、所有者名を付けずにシノニム名で参照します。ユーザーがシノニムと同名の表を所有している場合、DBMasterはシノニムを無視し常に表の方を使用します。表をシノニムで参照するには、シノニムに表の完全修飾名を与えます。表やビューを削除すると、表やビューのシノニムも全て自動的に削除されます。

シノニム名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例1**　ユーザー**User1**が所有する表**AllEmployees**にシノニム**AllEmp**を作成します。この後の**SQL**文では、完全修飾表名**User1.AllEmployeesの代わりにシノニムAllEmp**を使用します：

```
CREATE SYNONYM AllEmp FOR User1.AllEmployees;
```

➲ **使用例2**　ユーザー**User2**が所有するビュー**SalesEmployees**にシノニム**SalesEmp**を作成します。この後の**SQL**文では、完全修飾ビュー名**User2.SalesEmployees**の代わりにシノニムSalesEmpを使用します：

```
CREATE SYNONYM SalesEmp FOR User2.SalesEmployees;
```

関連SQL

CREATE TABLE

CREATE VIEW

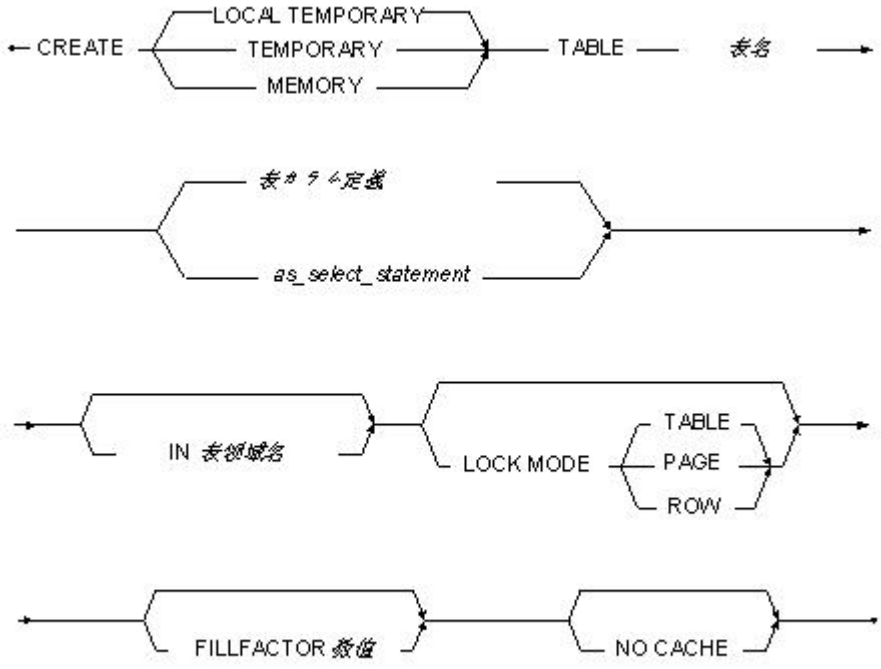
DROP SYNONYM

3.41 CREATE TABLE

目的

データベースに新規表を作成します。

構文



表名	作成する新規表の名前
表カラム定義	表カラムの定義
制約名	設定する制約の名前
条件式	真または偽と評価される式
主キーと一意定義	表キーと一意キーの定義
外部キー定義	外部キーの定義
表領域名	作成する表の表領域の名前
数値	フィルファクタのパーセント

説明

CREATE TABLE文は、表を新規に作成します。表を作成する際には表領域を指定します。初期設定では、表はシステム表領域に作成されます。RESOURCE以上のセキュリティ権限をもつユーザーがCREATE TABLEを実行することができます。

表は、リレーショナル・データベースの基本となるデータ・ストレージ単位です。データベースの入力データは全て表に格納されます。個々の表は、実世界のオブジェクトの一つのタイプを表し、そのタイプのオブジェクトの情報をもっています。オブジェクトには、顧客や製品のような実在オブジェクト、注文やトランザクションのような抽象オブジェクトがあります。データベース内の各表は一意な名前をもち、名前は、通常、表に格納されるオブジェクトのタイプを表します。表は、オブジェクトの情報を行とカラムに格納します。

行はレコードまたは組（タプル）とも呼ばれ、同じタイプの特性をもつ実体（エンティティ）の定義情報をもちます。各行は、そのタイプの実体の個々の出現を表します。各行は実体の一つ以上の特性を使用して識別されます。行は特定の順序をもちません。行が常に同じ順序でリストされる保証はありません。

カラムはフィールドまたは属性とも呼ばれ、実体の特性を定義する情報をもっています。各カラムは、実体の個々の出現データが格納される一つの特性または項目を表します。カラムは、カラムを表す名前とデータ型を使用して識別します。カラムには一意なカラム名を付けます。表内のカラムを並び替えても、SQL文には影響を与えません。

カラムに制約または規則を適用して、データの整合性を確保します。表を作成するときは、個々のカラムにドメイン制約やカラム整合性制約を適用し、また、表整合性制約を適用します。

ドメイン制約は、ドメイン定義の一部として定義され、ドメインを基にする全てのカラムに適用されます。新規の行を挿入したり既存の行を更新するときは、各カラムのドメイン制約が評価されます。ドメイン制約には、NULL／NOT NULL制約、初期設定値、CHECK制約があります。

カラム制約は、特定のカラムにだけ適用され、同じ表内の他のカラムには影響しません。新規の行を挿入したり既存の行を更新するときは、各カラム制約が評価されます。カラム制約には、NULLまたはNOT NULL制約、初期設定値、CHECK制約があります。

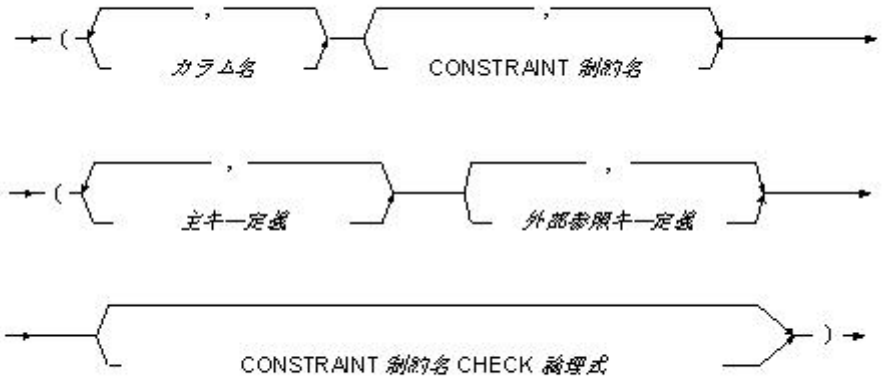
表制約は、カラムの組に対して定義されます。新規の行を挿入したり既存の行を更新するときは、全てのドメイン制約とカラム制約が真と評価された後に表制約が評価されます。表制約も真と評価されと、そのSQL文が処理されます。表制約には、UNIQUE制約とCHECK制約、主キー、外部キーがあります。

表を作成するときは、表名とカラム定義を与えることが最低限必要です。表は、1～2000カラムまでもつことができます、（表の最大カラム数はページサイズに依存します）。

表は、所有者名と表名の組み合わせ（完全修飾名と言う）で一意的に識別されます。表名の最大長は128字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。表はデータベース内で一意にします。DBA権限をもつユーザーは、表所有者名を指定して表を作成することができます。所有者名はデータベースのユーザー名を指定します。初期設定の表所有者は、表の作成者です。

カラム定義には、少なくとも1個のカラム名とデータ型またはドメインを与えます。カラム定義の構文とカラム定義に用いられるキーワードの使用法を次ページに示します。

カラム定義



カラム名	定義カラムの名前
データ型	カラムのデータ型名
ドメイン名	データ型の代わりに使用するドメイン名
リテラル値	値が挿入されないときに使用するリテラル値
定数	値が挿入されないときに使用する定数
関数名	値が挿入されないときに使用する組み込み関数
制約名	設定する制約の名前
条件式	真または偽と評価される式
主キーと一意キー定義	主キーと一意キーの定義
外部キー定義	外部キーの定義

表のカラムは、所有者名・表名・カラム名の組み合わせ（完全修飾名と言われる）によって一意的に識別されます。カラム名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。カラム名は表内で一意にします。

DBMasterは次のデータ型をサポートします。BIGINT、BIGSERIAL、BINARY、CHAR、DATE、DECIMAL（NUMERIC）、DOUBLE、FILE、

FLOAT、INTEGER、LONG VARBINARY（BLOB）、LONG VARCHAR（CLOB）、OID、SERIAL、SMALLINT、TIME、TIMESTAMP、VARCHAR。

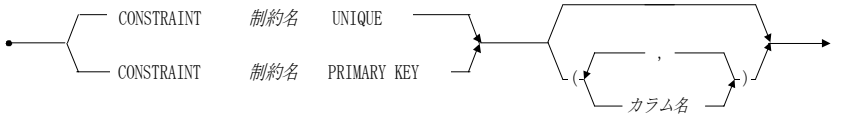
データ型の代わりにドメインを使用することができます。ドメインは、データ型、初期設定値、制約の組み合わせです。ドメインを使用してカラムを定義すると、これらがカラムに適用されます。初期設定値と制約については、DEFAULTとCHECKキーワードの説明を参照してください。ドメインの初期設定値と制約は、カラム定義で指定された初期設定値と制約によって打ち消されます。カラム定義でドメインの制約に制約を追加することもできます。

NULL／NOT NULLキーワードはオプションです。これらのキーワードは、新規行のカラムがNULL値を取れるかどうかを指定します。NULLキーワードはカラム値を未定にしてもよい指定、NOT NULLキーワードはカラム値を挿入しなければならない指定です。NULL／NOT NULLキーワードを指定しないと、初期設定としてNULLが使用されます。

DEFAULTキーワードはオプションです。このキーワードは、行のカラム値が空のときにカラムに挿入される初期設定値を指定します。初期設定値としては、リテラル値、定数、組み込み関数、NULLキーワードを指定することができます。組み込み関数は、PI( )、NOW( )、USER( )のように引数の無い関数を使用します。DEFAULT値をNULLにすると、カラムをNOT NULLに定義することはできません。

CHECKキーワードはオプションです。このキーワードは、カラムに入力できる値の範囲を指定します。真または偽と評価される任意の条件式で値の範囲を指定することができます。CHECK条件式の中では、カラム値を表すためにVALUEキーワードを使用することができます。CHECK条件が満たされないときは、そのSQL文は処理されません。

主キーと一意定義



制約名　　設定する制約の名前

カラム名　　主キーにするカラムの名前

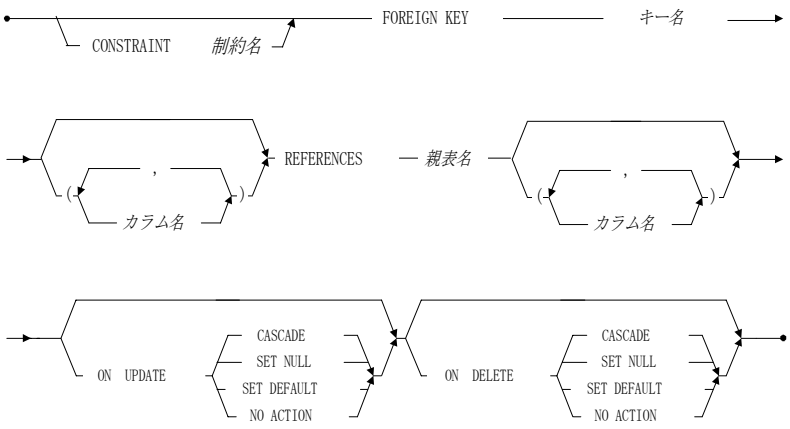
キーは、表内の特定の行の識別に役立てるカラムまたはカラムの組み合わせです。キーを構成するカラムをキーカラムといいます。一意キーとは、重複するキー値をもつレコードが無いキーのことです。

主キーは、表内の行を一意的に識別するキーです。主キーが無いと、行は重複値をもつ可能性があり、表内の行を他と区別することができなくなります。重複値をもつカラムを主キーに定義したり、既存の主キーに重複値を入力することはできません。

各レコードの主キーの値が一意でなければならないことから、主キーは表のデータ整合性を保証します。これは、主キーカラムが重複値およびNULL値を取れないこと、NOT NULLを用いて主キーカラムを定義すべきことを意味します。最大16カラムまでの主キーを構成することができます。ただし、主キーカラムの合計サイズは1024バイト以下でなければなりません。

各表には1つだけ主キーを定義することができます。主キーの名称変更はできません。DBMasterは、各表の主キーに対して、PrimaryKeyという名前の一意索引を自動的に作成し、内部的に管理します。主キーの索引は自動的に作成されます。問合せのパフォーマンスを上げるために、別の索引を主キーカラムに作成する必要はありません。

外部キー定義



制約名	設定する制約の名前
キー名	作成する外部キーの名前
カラム名	1. 作成する外部キーのカラムの名前
	2. 外部キーによって参照されるカラムの名前
親表名	外部キーによって参照される表の名前

外部キーは、別の表の主キーまたは一意索引に対応するキーです。二つの表は、共通のキーデータによって親子関係が付けられます。親表は主キーまたは一意索引をもち、子表は親表のカラムに対応する外部キーカラムをもちます。

参照整合性は、子キーの値に対応する親キーの値があることを保証します。参照整合性は、外部キーによって確立される親子関係を使用して表間に施行されます。DBMasterは、外部キーの定義を通して、自動的に表間の参照整合性制約をサポートします。子表にレコードを追加するときは、子キーの値が親キーに存在していなければなりません。親表からレコードを

削除するときは、対応する子キーをもつ全てのレコードを子表から削除しておかなければなりません。

ON UPDATE／ON DELETEキーワードはオプションです。これらのキーワードは、子キーによって参照される親キーの値を更新/削除するときに対応する全ての子キーの行に実行される参照アクションを指定します。参照アクションには、CASCADE、SET NULL、SET DEFAULT、NO ACTIONがあります。

CASCADEは、親キーを更新/削除するときに対応する子キーの値を更新/削除します。更新された子キーの値は親キーと同じ値になります。

SET NULLは、親キーを更新/削除するときに対応する子キーの値をNULLに設定します。子キーがNOT NULL制約を付けて定義されているときは、SET NULLアクションを指定することはできません。

SET DEFAULTは、親キーを更新/削除するときに対応する子キーの値をカラムの初期設定値に設定します。初期設定値がNULLであり、子キーがNOT NULL制約を付けて定義されているときは、SET DEFAULTアクションを指定することはできません。

NO ACTIONは、通常の参照整合性の規則に従います。初期設定の参照アクションはNO ACTIONです。

外部キーの個数には実用上の制限はありません。親キーは主キーまたは他の任意の一意索引ですが、子キーを追加するときは、事前に親キーを作成しておかなければなりません。親キーと子キーのカラム数とカラムタイプまたはカラム長は同じでなければなりません。両表の対応カラムの順序は違っていてもかまいませんが、外部キー定義では対応する順にリストします。初期設定親キーは、親表の主キーです。

外部キーカラムにはNULL値があってもかまいません。外部キーがNULL値のときは、自動的に参照整合性が満たされます。ビューに外部キーを作成することはできませんが、シノニムには作成することができます。外部キー名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

表オプション

DBMasterには、表作成時に指定することができるオプションのキーワードには、TEMPORARY、IN、CHECK、LOCK MODE、NOCACHE、FILLFACTORがあります。

TEMPORARYキーワードは、オプションです。替わりに、TEMP、LOCAL TEMPORARY、LOCAL TEMPを指定できます。これらのキーワードは、標準表ではない一時表を作成する指定です。一時表のデータ・アクセスは、ロックやジャーナル・レコードを使用しないので高速になります。一時表は、表所有者だけが使用することができ、データベースから切断すると自動的に削除されます。データベース接続中でも、DROP TABLE文を使用して削除することができます。

MEMROYキーワードはオプションです。 ほとんどすべての意図および目的に対して、メモリ表はDBMasterの通常の表として同じように機能します。 相違は、メモリ表が一時表であり、その寿命サイクルが接続ベースであるというところにあります。 これは、ユーザーがメモリ表を作成しているとき、ユーザーがメモリ表を削除すれば、またはユーザーがデータベースから切断されればメモリ表が削除されることを意味します。 通常の表とは異なり、メモリ表は作成された接続のメモリにのみ保存されます。 これらの接続は他の接続によっては使用されず、また選択または挿入されたデータのみ持つことが可能で、そのデータは更新または削除することができません。 メモリ表は、次のトランザクション制御をサポートします: セーブポイントのコミット、ロールバック、定義およびセーブポイントへのロールバック。

これらのキーワードは、表が標準表の代わりに一時表として作成される必要があることを指定します。 ロックがいっさい使用されず、またいかなるジャーナル・レコードも一時表に対して書き込まれないため、一時表ではデータアクセスが高速になります。 しかし、一時表は表の所有者しか使用できず、ユーザーがデータベースから切断すると自動的に削除されます。また、DROP TABLEコマンドを使用すれば、データベースに接続されて

いる間いつでも一時表を削除できます。

INキーワードはオプションです。このキーワードは、表を作成する表領域の名前を指定します。表領域は、データベース情報を管理可能な領域に区分けする論理ストレージ領域です。表を論理的にグループ化したり、頻繁

に使用する表を別々のストレージに置くことが可能になります。初期設定の表領域は、システム表領域です。

CHECKキーワードはオプションです。このキーワードは、カラム定義のCHECKキーワードと同様の振る舞いをします。通常は、複数のカラムのデータが受入れ可能な値の範囲にあることを保証するのに使用します。条件式は、真または偽と評価される任意の式です。条件式の中では、カラム名を使用してカラムの値を表すことができます。CHECK条件を満たさないSQL文は処理されません。

LOCK MODEキーワードはオプションです。このキーワードは、表データをアクセスするときに使用するロックレベルを指定します。DBMasterには、表、ページ、行の3つのロックモードがあります。初期設定はページロックモードです。表のロックモードは、SYSTABLEのLOCKMODEカラムを見て判定することができます。

LOCK MODE TABLEは、表全体をロックします。表ロックは、他のユーザーが表を同時並行アクセスするのを禁止するので同時実行性が低下します。しかし、使用するロックリソースは小さくなり、システム制御域(SCA)の必要メモリも小さくなります。

LOCK MODE PAGEは、データページ単位でロックします。このモードは同時実行性とロックリソースの双方を考慮した選択です。他のユーザーは、他のページをアクセスすることができるので中位の同時実行性が得られます。しかし、同じページ内のデータを同時アクセスすることはできません。

LOCK MODE ROWは、行単位でロックします。他のユーザーは、ロックされた行以外のデータを同時アクセスすることができるので同時実行性が増大します。しかし、使用するロックリソースは大きくなり、SCAの必要メモリも大きくなります。

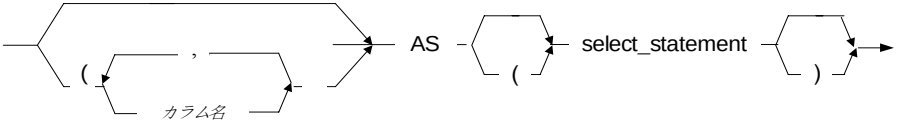
FILLFACTORは、データページの挿入パーセントを指定します。既存レコードの更新用スペースを空けておくことによって、データページ使用の最適化が可能になります。50～100の値を指定することができ、50%～100%の挿入を表します。表のFILLFACTORは、SYSTABLEシステム表のFILLFACTORカラムを見て判定することができます。

NOCACHEは、表検索に使用するデータキャッシュのページバッファを制限します。ページバッファは、直近に使用したページがバッファチェーンの先頭になり、最長未使用ページがバッファチェーンの最後になるように格納されます。NOCACHEオプションを指定すると、表検索で読み込まれたデータページはバッファチェーンの最後に置かれます。表検索は、バッファチェーンの最終バッファを掃き出してから開始され、検索中は後続のデータページが直前のページを上書きします。結果として、表検索に使用するページバッファは1ページバッファに制限されます。キャッシュモードは、SYSTABLEシステム表のCACHEMODEカラムを見て判定することができます。

表を作成すると、作成者が表所有者になります。表所有者は表の全てのオブジェクト権限をもち、他のユーザーにオブジェクト権限を与えることができます。セキュリティ権限がCONNECTに落とされても、表所有者としてのオブジェクト権限は残ります。

CREATE TABLE AS SELECT

CREATE TABLE AS SELECT構文を使い、表とカラム定義の作成とselect_statementによるデータのコピーを行います。CREATE VIEWのようにカラム定義を作成しSELECT INTOのようにデータをコピーします。



テーブル作成：as_select_statement構文

使用例

➲ **使用例1**　システム表領域に表**Scores**を作成します。表には、データ型**INTEGER**のカラム**StudentNo**、**Math**、**English**、**Science**、**History**があります：

```
CREATE TABLE Scores (StudentNo INTEGER,
                        Math INTEGER,
                     English INTEGER,
                     Science INTEGER,
                     History INTEGER);
```

➲ **使用例2**　使用例1と同じ表を表領域**StudentRecords**に作成します。表所有者は**Madison**です。カラム**Math**、**English**、**Science**、**History**は**NULL**値を禁止し、初期設定値**0**を設定します：

```
CREATE TABLE Madison.Scores
(StudentNo INTEGER NOT NULL,
                        Math INTEGER NOT NULL DEFAULT 0,
                     English INTEGER NOT NULL DEFAULT 0,
                     Science INTEGER NOT NULL DEFAULT 0,
                     History INTEGER NOT NULL DEFAULT 0)
            IN StudentRecords;
```

➲ **使用例3**　使用例2と同じ表を作成します。カラム**Math**、**English**、**Science**、**History**の値は**0**～**100**の間になければなりません：

```
CREATE TABLE Scores (StudentNo INTEGER NOT NULL,
                        Math INTEGER NOT NULL DEFAULT 0
                                  CHECK VALUE <= 100 AND VALUE >= 0,
                     English INTEGER NOT NULL DEFAULT 0
                                  CHECK VALUE <= 100 AND VALUE >= 0,
                     Science INTEGER NOT NULL DEFAULT 0
                                  CHECK VALUE <= 100 AND VALUE >= 0,
```

```
                        History INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0)
              IN StudentRecords;
```

➲ **使用例4**　使用例3と同じ表を作成します。カラム**Math**、**English**、**Science**、**History**の合計が**400**以下である表制約を定義します：

```
CREATE TABLE Scores (StudentNo INTEGER NOT NULL,
                           Math INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0,
                        English INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0,
                        Science INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0,
                        History INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0)
              IN StudentRecords
              CHECK Math + English + Science + History <= 400;
```

➲ **使用例5**　使用例4と同じ表を作成します。ロックモードを**PAGE**に設定し、**FILLFACTOR 90**を指定し、NOCACHEを指定します：

```
CREATE TABLE Scores (StudentNo INTEGER NOT NULL,
                           Math INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0,
                        English INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0,
                        Science INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0,
                        History INTEGER NOT NULL DEFAULT 0
                                        CHECK VALUE <= 100 AND VALUE >= 0)
              IN StudentRecords
              CHECK Math + English + Science + History <= 400
```

```
            LOCK MODE PAGE
            FILLFACTOR 90
            NOCACHE;
```

➲ 使用例6　以下は、Scoresテーブルから Math score > 70's StudentNo を選択するクエリからテーブルを作成します:

```
CREATE TABLE Scores70 AS SELECT StudentID, Math from Score where Math > 70)
            IN tablespace1;
```

関連SQL

ALTER TABLE ADD COLUMN

ALTER TABLE FOREIGN KEY

ALTER TABLE PRIMARY KEY

ALTER TABLE DROP FOREIGN KEY

ALTER TABLE DROP PRIMARY KEY

ALTER TABLE SET OPTIONS

CREATE INDEX

CREATE SYNONYM

CARETE TABLESPACE

CREATE VIEW

GRANT (オブジェクト権限)

REVOKE (オブジェクト権限)

LOCK TABLE

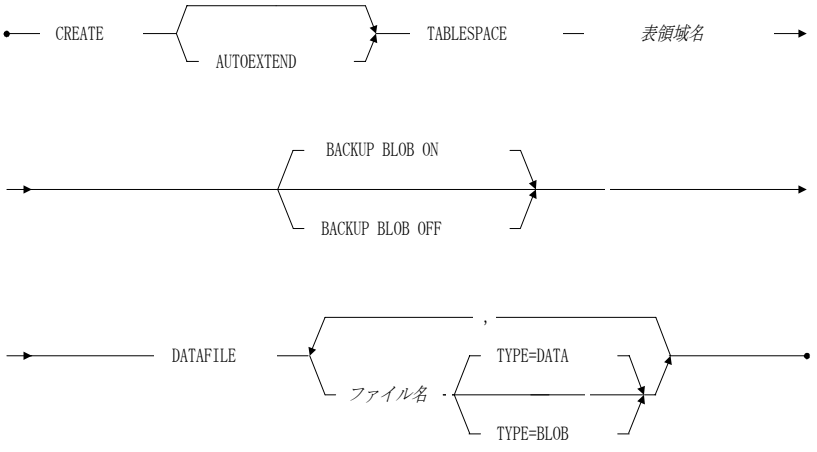
DROP TABLE

3.42 CREATE TABLESPACE

目的

標準または自動拡張の表領域を新規に作成します。

構文



表領域名	作成する表領域の名前
ファイル名	表領域ファイルの論理ファイル名

説明

CREATE TABLESPACE 文は表領域を新規に作成します。新規表領域は、データベースが使用することのできる物理ストレージを拡大します。DBAまたはSYSADMだけが実行することができます。

DBMasterは、リレーショナル・データモデルを使用して物理ストレージモデルの細部を隠蔽し、論理ストレージモデルを使用してデータを表現しま

す。物理ストレージモデルでは、データベースのデータを格納する物理ストレージ構造はファイルです。Unixのローデバイスを除いて、ファイルはオペレーティング・システムによって管理され、一方、ファイル内のデータはDBMSによって管理されます。DBMasterでは、3種類のファイル、データ、BLOB、ジャーナルを通常の操作に使用します。

データファイルとBLOBファイルには、ユーザーとシステムのデータを格納します。これらファイルは類似の特性をもっていますが、パフォーマンスを上げるために、データとBLOBは別々の方法で管理されます。データファイルには表と索引のデータを格納し、BLOBファイルにはバイナリ・ラージ・オブジェクト(BLOB)を格納します。

ジャーナル・ファイルは、データベースの全ての変更履歴と変更ステータスをリアルタイムに記録する特殊なファイルです。ジャーナルは、失敗したトランザクションによる変更のundo、成功した場合でもデータベース損傷によってディスクに書き込まれていない変更のredoを可能にします。ジャーナル・ファイルは、データベース管理システムのみによって使用され、ユーザーデータの格納には使用されません。

論理ストレージモデルは、データベース情報を管理可能なエリアに区分けするのに用いられる論理ストレージ構造は表領域です。表領域には、種々の表や索引が含まれます。表領域内のデータはDBMSによって管理されますが、物理的にはデータとBLOBファイルに格納されます。標準、自動拡張、システムの3種類の表領域があります。

標準表領域は、1個以上のデータファイルあるいはBLOBファイルをもつ固定サイズの表領域です。標準表領域は、既存ファイルを拡大したり新規ファイルを追加することによって手動で拡張することができます。標準表領域は、最大32767個のファイルをもつことができ、ファイルサイズの合計は最大8 TBです。UNIXプラットフォームでは、ローデバイス上に標準表領域を置くことができます。

注：　ローデバイスの詳細情報はUNIXシステムのマニュアルを参照してください。

自動拡張表領域は、必要に応じて自動的にサイズを拡大し追加データを格納する表領域です。自動拡張表領域は、少なくとも1個のデータファイル

をもたなければなりません。BLOBファイルをもたせることもできます。自動拡張表領域が一杯になっても、データベースは作動します。最大ファイルサイズは、8_TBまたはディスクの最大容量です。自動拡張表領域を手動で拡大するときは、既存のファイルを拡大します。自動拡張表領域に大量のデータを挿入するときは、事前にスペースを割当てておくとパフォーマンスが良くなります。ローデバイスを自動拡張表領域に使用することはできません。

システム表領域は、データベース作成時に自動的に生成されます。データベースにはシステム表領域が1個あり、システムカタログ表が置かれています。システムカタログ表は、データベース全体のスキーマ、セキュリティ、ステータス情報を格納します。システム表領域は、自動拡張表領域として作成されます。システム表領域には、システムデータを格納する1個のデータファイルと1個のBLOBファイルが自動的に作成されます。システム表領域は、標準表領域に変換することができます。ローデバイスを使用することはできません。

AUTOEXTENDキーワードはオプションです。このキーワードは、自動拡張表領域を作成する指定です。自動拡張表領域は、追加スペースが必要になると自動的にサイズを拡張します。自動拡張表領域は、いつでも標準表領域に変換することができ、逆に自動拡張表領域に戻すことができます。

BACKUP BLOBキーワードはオプションです。このキーワードは、データベースがBACKUP_DATA_AND_BLOBモードのときに表領域内のBLOBデータをバックアップするかどうかを指定します。BACKUP BLOB ONは、BACKUP_DATA_AND_BLOBモードのときに表領域内の全てのBLOBデータをバックアップします。BACKUP BLOB OFFは、バックアップ・モードが何であれ、表領域内のBLOBデータをバックアップしません。

データベースのデータ独立性を確保するために、データベース内では、オペレーティング・システムのファイルを直接参照することはしません。各データベース・ファイルには、物理ファイル名と論理ファイル名の2つの名前があります。物理ファイル名はオペレーティング・システムによって使用され、一方、論理ファイル名はデータベースによって使用される名前です。論理ファイル名は、**dmconfig.ini** ファイルで物理ファイル名に対応付けられます。CREATE TABLESPACEを実行する前に、**dmconfig.ini**のデ

ータベース構成セクションに論理ファイル名、物理ファイル名、物理ファイルの初期サイズを指定するエントリを作成します（使用例を参照）。

DATAFILEキーワードは、表領域を作成するときの論理ファイル名とファイルのタイプを指定します。表領域の種類とディスク容量が許す範囲で最大32767個のファイルを指定することができます。表領域作成には、少なくとも1個のデータファイルが存在しなければなりません。ALTER TABLESPACE文を使用して、表領域にファイルを追加することができます。

TYPEキーワードは、データファイルとBLOBファイルのどちらであるかを指定します。TYPE=DATAはデータファイルを作成し、TYPE=BLOBはBLOBファイルを作成します。TYPEキーワードでファイルのタイプを指定しないと、初期設定はデータファイルとして作成されます。

ディレクトリあるいはパスが指定されていない物理ファイルは、全て**dmconfig.ini**のDB_DBDIRキーワードで指定される初期設定のデータベースディレクトリに作成されます。初期ファイルサイズは、データファイルはデータページ数、BLOBファイルはBLOBフレーム数で指定します。

データファイルの初期ファイルサイズは、2-2,147,483,647の範囲のページ数を指定します。キロバイトに換算するには**dmconfig.ini**ファイルにDB_PGSIZの値 を乗算します。BLOBファイルの初期ファイルサイズは、2～524,287の範囲のフレーム数を指定します。キロバイトに換算するには**dmconfig.ini**ファイルのDB_BFRSZの値を乗算します。

表領域内の各ファイルは、別のディスクか同じディスクでも別のパスを指定します。同じディスク位置を指定してはいけません。UNIXの場合は、標準表領域のファイルをローデバイスに割当てることができます。ローデバイスを使用すると、通常のオペレーティング・システムファイルよりも高速にアクセスすることができ、パフォーマンスを改善します。オペレーティング・システムのコールを経由せずに、直接ローデバイスに書き出します。

表領域名と論理ファイル名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

物理ファイル名の最大長はドライブとパス名を含めて255字です。オペレーティング・システムで使用できる文字・記号を使用することができます。物理ファイル名の大文字と小文字を区別するかどうかは、オペレーティング・システムに依存します。

使用例

➲ **マップ1**　使用例1を実行する前に、論理ファイル名を物理ファイル名にマップし、初期物理ファイルサイズをデータファイルはページ数、**BLOB**ファイルはフレーム数で指定する行を**dmconfig.ini**ファイルに追加します。**datafile**サイズは**800KB**、**blobfile**サイズはフレームサイズ32KBを使用すると3200KBになります：

```
datafile = c:\DBMaster\database\ ts_reg _df.db 100
blobfile = c:\DBMaster\database\ ts_reg _bf.bb 100
```

➲ **使用例1**　論理データファイル名**datafile**と論理**BLOB**ファイル名**blobfile**をもつ標準表領域**ts_reg** を作成します。表領域には、最大**32767**までのデータファイルまたは**BLOB**ファイルを追加することができます：

```
CREATE TABLESPACE ts_reg DATAFILE datafile TYPE=DATA, blobfile TYPE=BLOB;
```

➲ **マップ2**　使用例2を実行する前に、論理ファイル名を物理ファイル名にマップし、初期物理ファイルサイズをデータファイルはページ数、**BLOB**ファイルはフレーム数で指定する行を**dmconfig.ini**ファイルに追加します。**datafile**サイズは**800KB**、**blobfile**サイズはフレームサイズ**32KB**を使用すると3**200KB**になります：

```
datafile = c:\DBMaster\database\ ts_ext _df.db 100
blobfile = c:\DBMaster\database\ ts_ext _bf.bb 100
```

➲ **使用例2**　論理データファイル名**datafile**と論理**BLOB**ファイル名**blobfile**をもつ自動拡張表領域**ts_ext**を作成します。自動拡張表領域には、データファイルや**BLOB**ファイルは追加できません：

```
CREATE AUTOEXTEND TABLESPACE ts_ext DATAFILE datafile TYPE=DATA,
                                          blobfile TYPE=BLOB;
```

関連SQL

ALTER DATAFILE

ALTER TABLESPACE

CREATE TABLE

DROP TABLESPACE

3.43 CREATE TEXT INDEX

目的

既存の表カラムのテキスト索引を新規に作成します。

説明

DBMasterでは2つのタイプの索引、シグネーチャ・テキスト索引あるいはインバーテッドファイル(IVF)テキスト索引を作成することができます。シグネーチャ・テキスト索引は、索引が構築されているカラムと同じ表領域に構築されます。IVF索引は個別のファイルに構築され、索引の大きさに対して高いパフォーマンスを示します。

CREATE TEXT INDEX文は、テキストカラムのテキスト索引を作成します。テキスト索引は、表全体を検索することなく、テキストカラムにあるワードを素早く見つける全文検索のパフォーマンスを上げるために使用します。表所有者、DBA、SYSADM、表のINDEX権限をもつユーザーだけが実行することができます。

テキスト索引は、単一、複数の語や句を含むテキストカラムの行へ高速にアクセスするための機構です。テキスト索引には、テキストカラムの全てのテキストが符号化・構造化されており、表から直接検索するよりも高速に検索することができます。テキスト索引を作成しても、そのオペレーションはユーザーには見えません。DBMSが全文検索のパフォーマンスを上げるために使用します。

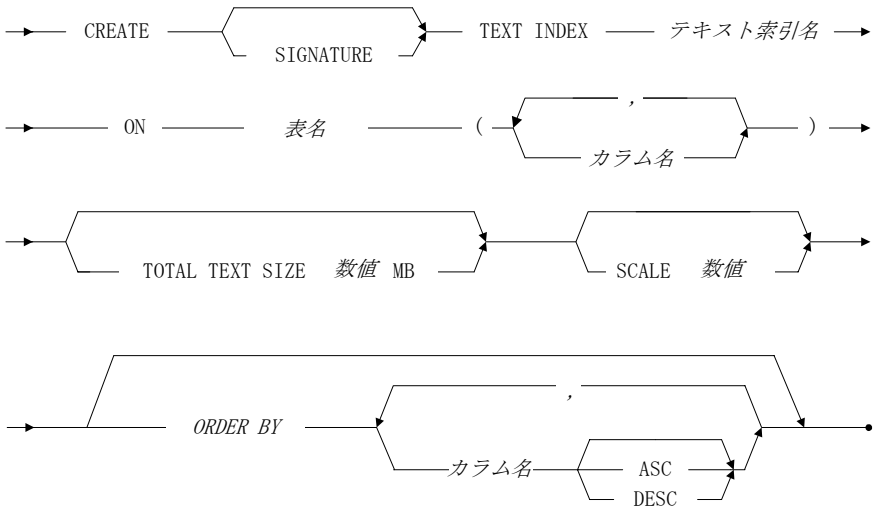
テキスト索引を作成する場合、索引名、表名およびカラムの名を指定してください。テキスト索引は、CHAR、VARCHAR、CLOB、NCHAR、NVARCHAR、NCLOBあるいはFILEデータ・タイプで定義されたカラム上で作成することができます。テキスト索引は、システム表、一時表あるいはビュー上で作成することができます。

Order By句は、語句検索に対応し、別のカラムに結果を並べます。Order Byカラムによるテキスト索引を作成した後に、DBMasterがテキスト索引上の問い合わせを処理している一方、結果はOrder Byカラム順で自動的に並べられて出力されます。例えば、カラム*content*を探索して*post time*カラムによってソートするためには、select文の終わりにOrder By post time句を加えてください。DBMasterは、Order By句の結果の上にソートを行わなければいけません。ソートには多くの時間かかるでしょう。*post time*カラムによるオーダーを備えたテキスト索引を作成したならば、句によるオーダーを加えずに、ソートされた結果を得ることができます。ASCかDESCキーワードで、表示が昇る順あるいは下る順であることを指定してください。デフォルト順序は昇る順です。Order Byカラム属性は rebuild indexコマンドのインクリメント部分に上に影響します。しかしながら、古いデータもしくはインクリメント・データを交えて再度レコードをソートすることはできません。

表にデータをロードする場合、DBMasterはその表上のテキスト索引を更新しません。可能な場合、表上のテキスト索引を作成する前にデータをすべてロードしてください。テキスト索引が作成された後で表に入力された箇所は、一致するテキストが含まれていても全文検索結果で返されないでしょう。探索結果にこれらの列を含むためには、REBUILD TEXT INDEXコマンドを使用して、テキスト索引を再建してください。

テキスト索引名は各表に対してユニークでなくてはなりません。テキスト索引名は、32文字の最大長を持っており、数字、文字、下線文字、およびシンボル$および#を含んでいるかもしれません。第1の文字は数字ではないかもしれません。

構文



テキスト索引名	作成するべきテキスト索引の名前
表名	テキスト索引を作成する表の名前
カラム名	索引を作成するカラムの名前
order byカラム名	スタートカラムの名前
数値	SCALE、TOTAL TEXT SIZEパラメータに使用する値

シグネーチャ・テキスト索引を作成するには、索引名、表名、カラム名を指定します。テキスト索引は、CHAR、VARCHAR、LONG VARCHAR、NCHAR、NVARCHAR 、NCLOB 、FILEデータ型のカラムに作成することができます。システム表、一時表、ビューには作成できません。表は多数のテキスト索引を持つことができ、多数のカラムにテキスト索引を建てることができます。

TOTAL TEXT SIZEはテキスト索引がMBで建てられるカラムのすべてのドキュメントの推定のサイズの合計です。範囲は1-200です。また、デフォルト価値は32です。この値はDBMasterによって評価および実行最適化のために使用され、カラムで許可されたドキュメントの数に現実に制約を置きません。推定のサイズの合計が200MBを超過する場合は、200MBを使用するか、あるいは著しく改善された問い合わせ実行のためのインバーテッドファイル(IVF)索引を作成してください。

SCALEはカラム・サイズを合計する索引サイズの予期された比率です。20にTOTAL TEXT SIZEをセットし、索引がおよそ10MBの記憶装置を使用することを期待すれば、50(50%)に規模をセットするべきです。規模がより大きいほど、探索実行はよりよいです。10-200からの範囲を入力することができます。デフォルト価値は40です。

➲ 使用例 1

下記は、すべてのパラメーターに対するデフォルト価値を使用して、**Employeesinfo**表の**FName**カラム上の**TxtIdx**と命名されるシグネ―チャ・テキスト索引、およびorder by **Emp_ID** カラムを作成します。

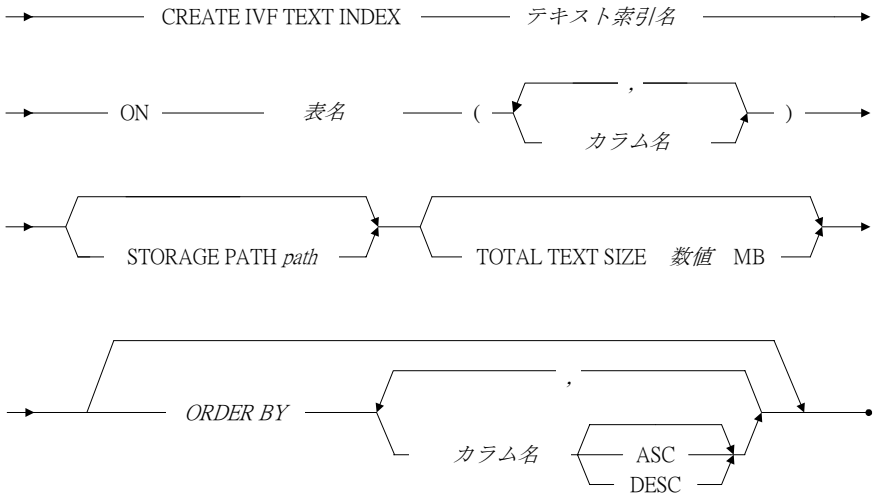
```
CREATE SIGNATURE TEXT INDEX TxtIdx ON Employeesinfo(FName) Order By Emp_ID
```

➲ 使用例 2

下記コマンドは、**Employeesinfo**表の**FName**カラム上の**TxtIdx**と命名されるシグネーチャ・テキスト索引を作成し、カラムのサイズの合計を20MBと見積もって、実際のテキスト索引のサイズの50%の索引を作成します。

```
CREATE SIGNATURE TEXT INDEX TxtIdx ON Employeesinfo (FName) TOTAL TEXT SIZE
20 MB scale 50
```

構文



テキスト索引名	作成するテキスト索引の名前
表名	テキスト索引を作成する表の名前
カラム名	索引を作成するカラムの名前
path	索引を格納するためのフル・ディレクトリ・パス
order byカラム名	スタートするカラムの名前
数値	TOTAL TEXT SIZEパラメータと使用する値

CREATE IVF TEXT INDEXコマンドは、指定されたカラム上の新しいインバーテッド・ファイル(IVF)テキスト索引を作成します。IVFテキスト索引は問い合わせのパフォーマンス向上のため、特に200MB以上のデータを含んでいるカラム上で、標準の索引の代わりに使用することができます。.

DBAもしくはSYSADM権限を持った、表所有者あるいはユーザは、IVFテキスト索引を作成することが可能です。

IVF索引はオペレーティング・システムのファイル・システムの中でソートされ、データベースを通って処理されます。索引が作成される場合、IVF索引が格納されるべき位置が指定されます。DBMasterは、IVF索引・ルート・ディレクトリ内のサブディレクトリの生成を管理します。

➲ 例 1

下記は、**Employeesinfo**表の**LName**カラム上の**TxtIdx**と命名され、すべてのパラメーターに対してデフォルト価値を使用するIVFテキスト索引を作成します。

```
CREATE IVF TEXT INDEX TxtIdx ON Employeesinfo (LName)
```

➲ 例 2

下記コマンドは、**Employeesinfo**表の**LName**カラム上の**TxtIdx**と命名されるIVFテキスト索引を作成し、100MBとカラムのサイズの合計を見積もっている間、論理的なファイルDB_IVFDIRにIVFテキスト索引を格納します。

```
CREATE IVF TEXT INDEX TxtIdx ON Employeesinfo (LName) STORAGE PATH DB_IVFDIR
TOTAL TEXT SIZE 100 MB ORDER BY Emp_ID ASC
```

関連SQL

CREATE INDEX

CREATE TABLE

DROP TEXT INDEX

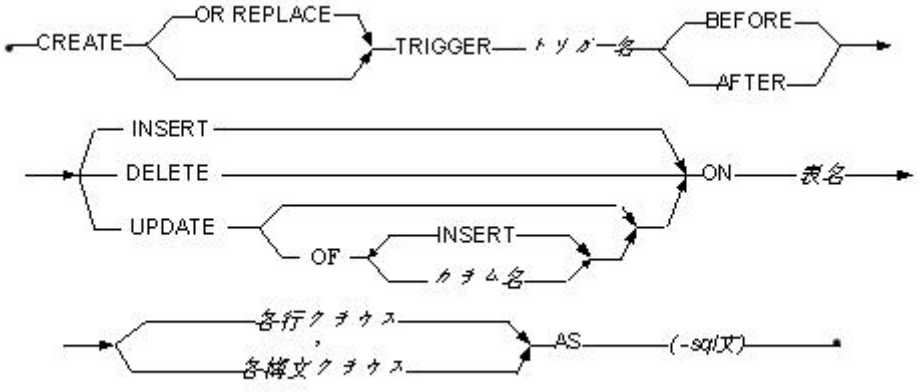
REBUILD TEXT INDEX

3.44 CREATE TRIGGER

目的

表のトリガーを新規に作成します。

構文



OR REPLACE	OR REPLACEで既存なトリガーを再び作成します。この句を使用して既存なトリガーの定義を変更できます
トリガー名	作成するトリガーの名前
カラム名	トリガーを作成するカラムの名前
表名	トリガーを作成する表の名前
sql_文	トリガーが起動したときに実行する *SQL* 文

説明

CREATE TRIGGER文は表のトリガーを作成します。トリガーは、標準的なSQL文を用いては不可能な方法でデータベースをカスタマイズするのに使用します。このSQL文は、トリガーアクションのSQL文を実行するのに必要な全てのセキュリティ権限とオブジェクト権限をもつ表所有者、DBA、SYSADMだけが実行することができます。

トリガーは、特定のイベントに応じて自動的に事前に定義されたSQL文を実行するデータベース・サーバーのメカニズムです。トリガーは、複雑で非類型的なデータベース操作を実行可能にします。トリガーはデータベース・サーバーの制御下にあり、起動の原因に関わらず整合性を保ってデー

タを処理することを保証します。トリガーはユーザーには見えずに起動します。

トリガーを作成するには、トリガー名、トリガー・アクションタイム(トリガーの起動はトリガー・イベントの前か後か)、トリガー・イベント(トリガーを起動させるイベント)、トリガー表(トリガーを作成する表)、トリガー・タイプ(トリガーの起動タイプ)、トリガー・アクション(トリガーが起動したときに実行するデータベースのアクション)を指定します。表を削除すると、自動的に表に作成されたトリガーも削除されます。

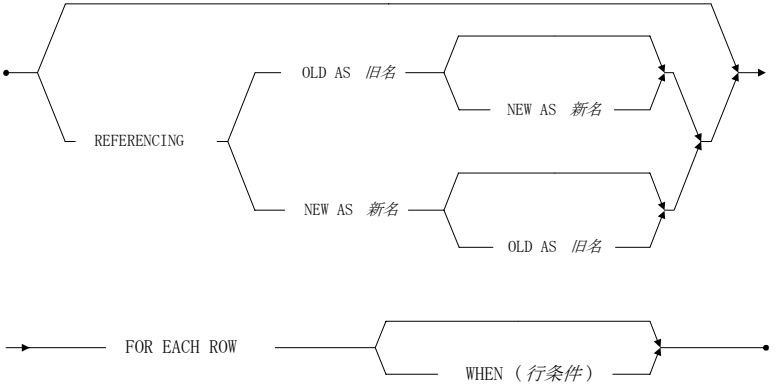
トリガーは表に関連付けられており、完全修飾名は使用しません。同じ表上の全てのトリガー名は、一意でなければなりません。トリガーアクションは、トリガー表所有者のセキュリティ権限とオブジェクト権限で動作します。トリガーイベントを起動するユーザーの権限で動作するのではありません。

BEFORE／AFTERキーワードは、トリガー・イベントの前／後の関係でトリガー・アクションの実行時点(トリガー・アクションタイム)を指定します。BEFOREキーワードは、トリガー・イベントの前にトリガー・アクションを実行する指示です。AFTERキーワード、トリガー・イベントの後にトリガー・アクションを実行する指示です。

INSERT／DELETE／UPDATEキーワードは、トリガーを起動するイベントを指定します。INSERTキーワードは表への行の挿入時のトリガー起動を指示し、DELETEキーワードは表からの行の削除時のトリガー起動を指示します。UPDATEキーワードは、表の任意のカラムの更新時のトリガー起動を指定します。特定のカラムの更新時のトリガー起動を指示するには、UPDATE OFを使用してカラムリストを指定します。UPDATE OFで指定したカラムリストは、UPDATEトリガーを起動するカラムを制限します。

ONキーワードは、トリガー表：トリガーが定義される表の名前を指定します。トリガー表は標準表でなければなりません。一時表、ビュー、シノニムにトリガーを作成することはできません。各トリガーには1個のトリガー表を指定します。

For Each Row 句



旧名	トリガー・アクション起動前のトリガー表の値を参照する別名
新名	トリガー・アクション起動後のトリガー表の値を参照する別名
行条件	トリガーを起動させる行の条件

REFERENCINGキーワードは、OLD、NEWキーワードの別名を指定します。行トリガーを作成するときは、トリガーアクションの中でトリガー起動の前／後のカラム値をOLD／NEWキーワードで参照します。しかし、OLD、NEWキーワードと同名の表OLDやNEWが存在するときは、REFERENCINGキーワードで付けた別名の方を使用します。

FOR EACH ROWキーワードは、行が変更される毎にトリガーを起動する指定です。FOR EACH ROWキーワードを指定した行トリガーは、トリガーを起動させる文が何も行を処理しなければ一度も起動しません。

WHENキーワードは、トリガーを起動するときの行条件を指定します。行条件は、行が変更される毎に評価されます。行条件が真ならばトリガーを起動し、偽ならば起動しません。WHEN条件の結果はトリガーアクションにのみ働き、トリガーイベントには何の影響も与えません。

For Each Statement句

FOR EACH STATEMENT

FOR EACH STATEMENTキーワードは、トリガーを起動するSQL文に1回だけ起動するトリガーの指定です。FOR EACH STATEMENTキーワードを指定した文トリガーは、トリガーを起動する文が何も行を処理しないときでも起動します。

トリガーが起動したときに実行する文をトリガー・アクションといいます。トリガー・アクションに指定できるのは、INSERT、UPDATE、DELETE、EXECUTE PROCEDURE文です。トリガー・アクションで組み込み関数を使用するときは、PI( )、NOW( )、USER( )のように引数の無いものだけを使用します。ストアド・プロシージャは、COMMIT、ROLLBACK、SAVEPOINTのようなトランザクション制御文を含むことができません。

トリガー表のイベントには、アクション・タイムBEFORE／AFTERとトリガー・タイプFOR EACH ROW／FOR EACH STATEMENTを組み合わせたトリガーを複数作成することができます。例えば、INSERTイベントには4組のトリガー、BEFORE／FOR EACH STATEMENT、BEFORE／FOR EACH ROW、AFTER／FOR EACH ROW、AFTER／FOR EACH STATEMENTを作成することができます。UPDATEとDELETEイベントにも同じ組み合わせのトリガーを作成することができます。

UPDATE OFを使用すると、各カラムに4組のUPDATE OFトリガー、BEFORE／FOR EACH STATEMENT、BEFORE／FOR EACH ROW、AFTER／FOR EACH ROW、AFTER／FOR EACH STATEMENTを作成することができます。表にUPDATE OFトリガーがあるときは、同じ表にUPDATEトリガーを作成することはできません。

トリガー名は表毎に一意にします。トリガー名の最大長は128字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ 使用例1　**Employeesinfo**表の更新前後の値を**NameChange**表に挿入する**UPDATE**トリガー**Trig_update**を作成します。トリガーは、更新された行毎にカラムには無関係にトリガー・アクションの前に起動します：

```
CREATE TRIGGER Trig_update BEFORE UPDATE ON Employeesinfo
                    FOR EACH ROW
                   (INSERT INTO NameChange
                         VALUES (OLD.FName, OLD.LName,
                                 NEW.FName, NEW.LName)
```

➲ 使用例2　**Employeesinfo**表に行が挿入されたときにストアド・プロシージャ**SendMail**を実行する**INSERT**トリガー**Trig_insert**を作成します。トリガーは、挿入された行毎にトリガー・アクションの後に起動します。**OLD**、**NEW**に別名を与えるために**REFERENCING**キーワードを使用します：

```
CREATE TRIGGER Trig_insert AFTER INSERT ON Employeesinfo
                    REFERENCING OLD AS pre NEW AS post
                    FOR EACH ROW
                   (EXECUTE PROCEDURE SendMail(pre.FName,
                                               pre.LName,
                                               WelcomeMessage);
```

➲ 使用例3　**Orders**表を更新するときに**LogTime**ストアド・プロシージャを実行する**UPDATE**トリガー**Trig_update**を作成します。トリガーは、更新される行には無関係に、トリガー・アクションの前に一度だけ起動します：

```
CREATE TRIGGER Trig_update BEFORE UPDATE ON Orders
                    FOR EACH STATEMENT
                   (EXECUTE PROCEDURE LogTime);
```

➲ 使用例4　データベースに**tb_staff**と**tb_change**表があることを仮定します：

```
CREATE TABLE tb_staff (FName char(10), LName char(10));

CREATE TABLE tb_change (new_FName char(10), new_LName char(10), old_FName char(10), old_LName char(10));
```

トリガーtrig_updateを作成します：

```
CREATE OR REPLACE TRIGGER trig_update BEFORE UPDATE ON tb_staff

          FOR EACH ROW   (INSERT INTO tb_change VALUES (NEW.FName, NEW.LName,

OLD.FName, OLD.LName));
```

関連SQL

ALTER TRIGGER ENABLE

ALTER TRIGGER REPLACE

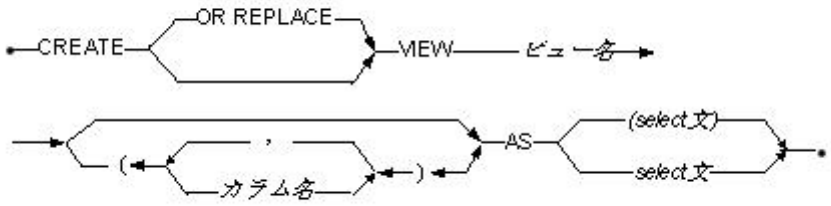
DROP TRIGGER

3.45 CREATE VIEW

目的

既存の表またはビューを元にして新規ビューを作成します。

構文



OR REPLACE	OR REPLACEで既存なビューを再び作成します。この句を使用して既存なビューの定義を変更できます
ビュー名	作成するビューの名前
カラム名	ビューのカラムの名前
select_文	ビューの内容を指定する SELECT 文

説明

CREATE VIEW文は、既存の表またはビューを基にしてビューを作成します。RESOURCE権限をもつ元表の所有者、表のビューまたはSELECT権限をもつユーザーだけが実行することができます。

ビューは既存の表やビューを元にする仮想表です。ユーザーには、ビューはカラム名とデータ行をもつ実表と同じように見えますが、実表のようにデータベースに永続的に格納されるものではありません。ビューを通して見るデータは、元の表のデータです。ビューデータとして物理的にデータベースに格納されてはいません。データベースに格納されるのは、ビュー名とビュー定義です。ビュー定義は、元の表からデータをアクセスするSQL問合せ文です。

ビューは、データベースの外観をユーザー要求に合せた個人的ビューを各ユーザーに与えるために使用します。認可されたデータを見ることだけをユーザーに許し、データのセキュリティとアクセス制限が設定されます。

ビューは、データベースの基礎構造からユーザーを分離します。ビューは、元の表が変更されても首尾一貫したイメージをユーザーに与えます。

ビューは、複数の表にある関連データを結合・グループ化して一つの表として表すことによって、データベースの構成を簡明にします。ビューは、結果セットに条件を付けることによって、元の表に格納されている行の部分セットを与えるのにも使用されます。

実表の代わりにビューを使用するときには、2つの不利な点、パフォーマンスと更新制限があります。ビューの問い合わせは、元の表に直接問い合わせるよりもパフォーマンスが悪くなります。最初にビュー定義を取り出し、通常の問合せに構築し直し、問合せを実行し、結果を表示します。また、ビューの使用には更新制限が課せられます。データベースは、複雑なビューによる更新を管理することができません。

ビュー定義のSELECT文は、INTOをもつことができません。現在は、単一の表を元にするビューだけが更新可能です。

ビューにはカラム名リストを指定します。カラム名の個数は、SELECT文のカラムの個数と一致しなければなりません。カラム名リストを指定しないと、元の表のカラム名がビューのカラムに継承されます。

ビュー名とカラム名の最大長は32字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

使用例

➲ **使用例　Employeesinfo**表のビュー**View_Emp**を作成します：

```
CREATE VIEW View_Emp AS SELECT Name, Salary from Employeesinfo WHERE Salary > 50000;
```

ビュー**View_Emp**を代わります：

```
CREATE OR REPLACE VIEW View_Emp AS SELECT Name, Salary from Employeesinfo WHERE Salary >= 100000;
```

関連SQL

CREATE SYNONYM

CREATE TABLE

CREATE VIEW

DROP VIEW

GRANT (オブジェクト)

REVOKE (オブジェクト権限)

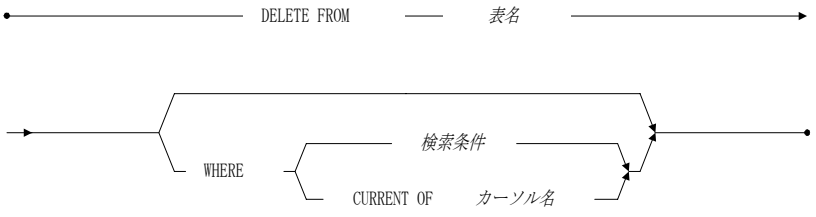
SELECT

3.46 DELETE

目的

表から行 (複数) を削除します。

構文



表名	行を削除する表の名前
検索条件	削除される行が満たすべき条件
カーソル名	位置付けDELETEに使用するカーソルの名前

*注：　カーソルは*ODBC *プログラム内でのみ使用可能です。*

説明

DELETE文は、検索条件を満たす全ての行を表から削除します。一つの表の行だけが削除されます。システム表の行を削除することはできません。このSQL文は、表所有者、DBA、SYSADM、表のDELETE権限をもつユーザーだけが実行することができます。検索条件を満たす行だけが削除されます。

*注：　検索条件の詳細情報については、*SELECT*文の*WHERE*句を参照してください。*

使用例

➲ **使用例1　Employeesinfo**表から従業員番号**1234**の従業員を削除します：

```
DELETE FROM Employeesinfo WHERE Emp_ID = '1234';
```

➲ **使用例2　Employeesinfo**表から従業員名が"**John**"で始まる従業員を全て削除します：

```
DELETE FROM Employeesinfo WHERE FName LIKE 'John%';
```

関連SQL

INSERT

LOCK TABLE

SELECT

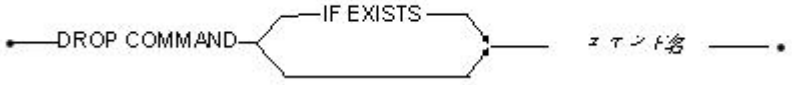
UPDATE

3.47 DROP COMMAND

目的

データベースからストアド・コマンドを削除します。

構文



IF EXISTS　　*コマンドが存在しない場合エラーを起こさないことを確保します*

コマンド名　　削除するストアド・コマンドの名前

説明

DROP COMMAND文は、ストアド・コマンドをデータベースから削除します。このSQL文は、ストアド・コマンドの所有者、DBA、SYSADMだけが実行することができます。

ストアド・コマンドは、実行形式のコンパイル済みSQLデータ操作文であり永続的にデータベースに格納されます。SQLデータ操作文をコンパイル・最適化せずに、繰り返しストアド・コマンドを実行することができます。ストアド・コマンドはストアド・プロシージャと似ていますが、一つのSQLデータ操作文だけからなり、プログラムロジックをもたない点が異なります。

ストアド・コマンドは、ストアド・コマンドが参照する表やカラムを削除する、カラム定義を変更する、BEFOREやAFTERキーワードを使用してカラムを追加すると不正になり、使用できなくなります。表にカラムを追加するときにBEFOREとAFTERキーワードを使用しなければ、ストアド・コマンドには何の影響もありません。不正のストアド・コマンドは、削除してデータベースから削除します。

使用例

➲ **使用例**　以下の構文を使用してストアド・コマンド**sc_select**を削除します：

```
DROP COMMAND sc_select;
```

或いは

```
DROP COMMAND IF EXISTS sc_select ;
```

関連SQL

CREATE COMMAND

EXECUTE COMMAND

GRANT (実行権限)

3.48 DROP DATABASE LINK

目的

データベースからデータベースリンクを削除します。

構文

リンク名　　データベースから削除するリンクの名前

説明

DROP DATABASE LINK文は、データベース・リンクをデータベースから削除します。DBA、SYSADM、リンク所有者だけがこのSQL文を実行することができます。また、*DBA*、*SYSADM*のみがパブリック・リンクを削除できます。

データベースリンクは、リモートデータベース接続を作成しリモートデータをアクセスできるようにします。リンクにはセキュリティ情報をもたせる利点があり、ローカルユーザー名とは別のユーザー名でリモート・デー

タベースに接続したり、アカウントのないパブリック・リンクを用いてリモート・データベースに接続することができます。

PUBLIC／PRIVATEキーワードはオプションです。これらのキーワードは、削除するリンクのタイプ、パブリックまたはプライベートを指定します。パブリック・リンクは、データベースの全ユーザーが使用することができます。プライベート・リンクは、リンク作成者だけが使用することができます。リンクのタイプを指定しないと、初期設定ではプライベート・リンクを削除します。

使用例

➲ **使用例1**　プライベート・リンク**FieldLink**を削除します：

```
DROP PRIVATE DATABASE LINK FieldLink;
```

➲ **使用例2**　パブリック・リンク**FieldLink**を削除します：

```
DROP PUBLIC DATABASE LINK FieldLink;
```

関連SQL

CREATE DATABASE LINK

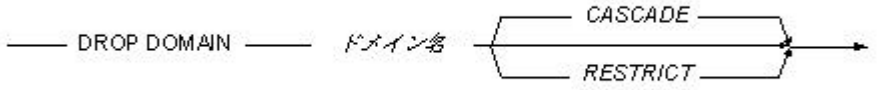
CLOSE DATABASE LINK

3.49 DROP DOMAIN

目的

データベースからドメインを削除します。

構文



ドメイン名　　データベースから削除するドメインの名前

説明

DROP DOMAIN文は、データベースに定義されているドメインを削除します。ドメインの所有者、DBA、SYSADMだけが実行することができます。

ドメインは、データ型、初期設定値、カラム制約をまとめたユーザー定義データ型です。カラムに挿入できる値の範囲を定義するために、CREATE TABLEやALTER TABLE ADD COLUMN文のカラム定義でデータ型の代わりにドメインを使用します。

ドメインで定義されたカラムが表にある間は、ドメインを削除することはできません。ドメインを削除するときは、最初にドメインを参照する全てのカラムを削除します。このためには、表を削除しドメインを使用せずに表を再作成するか、ALTER TABLE DROP COLUMN文を使用してカラムを削除します。

CASCADE/RESTRICTのキーワードはオプションです。これらのキーワードは、削除されたドメインを参照する従属したオブジェクトを削除するか、確認するかを指定します。 CASCADEのキーワードが指定されると、ドメインと共にすべての従属するオブジェクトが削除され、カラム定義がドメイン定義に置き換えられます。RESTRICTキーワードが指定されると、テーブル定義で参照されているドメインは削除されません。**RESTRICT**のキーワードは従属するオブジェクトがないドメインのみを削除します。

使用例

➲ **使用例**　ドメインValidDateを削除します：

```
DROP DOMAIN ValidDate;
```

関連SQL

CREATE DOMAIN

CREATE TABLE

ALTER TABLE ADD COLUMN

ALTER TABLE DROP COLUMN

DROP TABLE

3.50 DROP GROUP

目的

データベースからグループを削除します。

構文

グループ名　　データベースから削除するグループの名前

説明

DROP GROUP文は、データベースに定義されているグループを削除します。このSQL文は、DBA、SYSADMだけが実行することができます。

グループは、多数のユーザーをもつデータベースのオブジェクト権限の管理を容易にします。グループは、同じオブジェクト権限を必要とするユーザーをまとめるのに使用します。グループに与えられたオブジェクト権限は、自動的にグループのユーザー全員に与えられます。DBMasterでは、グループを更に別のグループのメンバーにする入れ子グループも使用できます。但し、入れ子のグループが自分のグループに循環してはいけません。

グループを削除すると、グループのユーザーは、そのグループの権限を失います。メンバーに直接与えられている権限、あるいは他のグループによって与えられている権限は残ります。PUBLICグループは削除することができません。PUBLICグループは、DBMasterによって内部的に管理されます。

使用例

➲ **使用例**　データベースからグループ**Manager**を削除します：

```
DROP GROUP Manager;
```

関連SQL

CREATE GROUP

ADD TO GROUP

REMOVE FROM GROUP

3.51 DROP INDEX

目的

表の索引を削除します。

構文

索引名	削除する索引の名前
表名	索引を削除する表の名前

説明

DROP INDEX文は、表の索引をデータベースから削除します。このSQL文は、表所有者、DBA、SYSADM、表のINDEX権限をもつユーザーだけが実行することができます。

索引は、キーと呼ばれるカラム(複数)の値から特定の表行を素早くアクセスできるようにする機構です。索引には表のキーカラムと同じデータがありますが、データは高速検索が可能なように構造化されソートされています。索引を作成しても、そのオペレーションはユーザーには見えません。DBMSは、可能な限り索引を使用して問合せのパフォーマンスを良くします。

システム表以外の任意の表の索引を削除することができます。索引を参照する外部キーがあるときは、索引を削除する前に、外部キーを削除しておきます。断片化して効率が悪くなった索引は、削除して再作成します。索引を再作成すると、稠密で断片化していない索引が作成されます。

使用例

➲ **使用例**　表**Employeesinfo** から索引**NameIndex**を削除します；**NameIndex**を参照する外部キーがあるときは、**NameIndex**を削除する前に外部キーを削除します：

```
DROP INDEX NameIndex FROM Employeesinfo;
```

関連SQL

CREATE INDEX

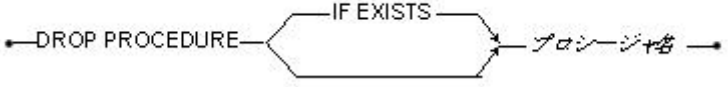
REBUILD INDEX

3.52 DROP PROCEDURE

目的

データベースからプロシージャを削除します。

構文



IF EXISTS	*プロシージャが存在しない場合エラーを起こさないことを確保します*
プロシージャ名	*データベースから移動するプロシージャの名*

説明

DROP PROCEDURE文は、表の索引をデータベースから削除します。このSQL文は、表所有者、DBA、SYSADM、表のPROCEDURE権限をもつユーザーだけが実行することができます。

使用例

➲ **使用例**　以下の構文を使用してプロシージャ***sp_proc1***を削除します：

```
DROP PROCEDURE sp_proc1;
```

或いは

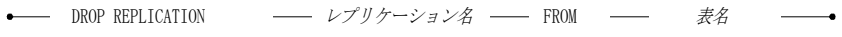
```
DROP PROCEDURE IF EXISTS sp_proc1;
```

3.53 DROP REPLICATION

目的

表からレプリケーションを削除します。

構文



レプリケーション名	削除するレプリケーションの名前
表名	レプリケーションを削除する表の名前

説明

DROP REPLICATION文は、データベースからレプリケーションを削除します。表所有者、DBA、SYSADMだけが実行することができます。

レプリケーションは、表全体または一部のカラムを遠隔地にコピーします。ターゲットのユーザーは、コピーされたデータで作業できるようになります。レプリケートされたコピーは、他の場所のデータベースと同期が取られます。各データベースは、低速ネットワーク接続を経由して別のマシンに行かずに、直接効率的にデータ要求に応じることができるようになります。レプリケーションは、ターゲット側にデータベースのバックアップを取るのとは異なり、ユーザーの介入無しにDBMS自身によってトランザクション毎に同期が取られます。

レプリケーションには、同期と非同期があります。同期レプリケーションは、ソース表の変更と同時にターゲット表を変更します。非同期レプリケーションは、ソース表の変更を格納しておき、予め定義されたスケジュールに従ってターゲット表を変更します。DROP REPLICATIONは、同期および非同期のレプリケーションを削除します。

使用例

➲ **使用例**　表**Employeesinfo**からレプリケーション**EmpRep**を削除します：

```
DROP REPLICATION EmpRep FROM Employeesinfo;
```

関連SQL

ALTER REPLICATION ADD REPLICATE

ALTER REPLICATION DROP REPLICATE

ALTER SCHEDULE

CREATE REPLICATION

CREATE SCHEDULE

DROP SCHEDULE

RESUME SCHEDULE

SUSPEND SCHEDULE

3.54 DROP SCHEDULE

目的

非同期レプリケーション用のスケジュールを削除します。

構文

ターゲット・データベース名　レプリケーション・スケジュールを削除するターゲット・データベースの名前

説明

DROP SCHEDULE文は、ターゲット・データベースのレプリケーション・スケジュールを削除します。レプリケーション・スケジュールを削除する前に、ターゲット・データベースにレプリケートする全ての非同期レプリケーションおよび関連する全ての非同期レプリケーションを削除しておきます。このSQL文は、ソース表の所有者、DBA、SYSADMだけが実行することができます。

ターゲット・データベース名の代わりにデータベースリンク名を指定することはできません。

使用例

➲ **使用例**　ターゲット・データベース**DivOneDb**のスケジュールを削除します：

```
DROP SCHEDULE FOR REPLICATION TO DivOneDb;
```

関連SQL

ALTER SCHEDULE

CREATE SCHEDULE

DROP REPLICATION

RESUME SCHEDULE

SUSPEND SCHEDULE

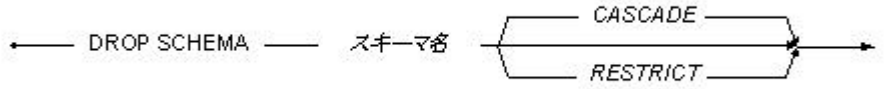
SYNCHRONIZE SCHEDULE

3.55 DROP SCHEMA

目的

DROP SCHEMAコマンドは、現在のデータベース・システムからスキーマを削除します。

構文



スキーマ名 ..... 削除されるスキーマの名前

説明

DROP SCHEMAコマンドは、現在のデータベース・システムからスキーマを削除します。スキーマは基本的には名前領域です: スキーマ・オブジェクトとしても知られている命名されたオブジェクト(表、ビュー、索引、シノニム、トリガー、ドメイン、コマンド、プロシージャ)を含みますが、その名前は他のスキーマに存在するほかのオブジェクトの名前に重複す

ることがあります。 スキーマ・オブジェクトは、接頭辞としてスキーマ名を持つそれらのオブジェクトの名前を修飾することによってアクセスできます。

スキーマを作成したユーザーまたはDBA権を有するユーザーのみが、データベースからスキーマを削除できます。削除するデータファイルは空でなければなりません。 スキーマ・オブジェクトを含むスキーマは、削除できません。 スキーマの削除を試みる前に、スキーマに含まれるすべてのスキーマ・オブジェクトを削除してください。

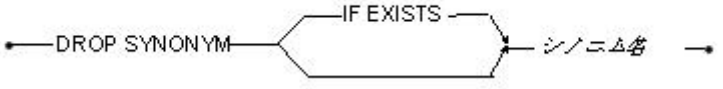
CASCADE/RESTRICTのキーワードはオプションです。これらのキーワードは、削除されるスキーマで参照される関連したオブジェクトを削除するか、確認するかを指定します。**CASCADE**のキーワードが指定されると、スキーマに従属するオブジェクトはすべて削除されます。RESTRICTのキーワードが指定されると、従属するオブジェクトのないスキーマだけが削除できます。

3.56 DROP SYNONYM

目的

データベースからシノニムを削除します。

構文



IF EXISTS	*シノニムが存在しない場合エラーを起こさないことを確保します*
シノニム名	データベースから削除するシノニムの名前

説明

DROP SYNONYM文は、表やビューのシノニムを削除します。シノニムの所有者、DBA、SYSADMだけが実行することができます。

表やビューは、通常、所有者名とオブジェクト名を複合した完全修飾名によって識別されます。シノニムは、表やビューの完全修飾名の使用を簡明にする助けをします。

シノニムは、表名やビュー名の代わりに使用することができる別名です。シノニムは、システムカタログ上のシノニム定義以外にはストレージを消費しません。シノニムを使用することによって、完全修飾名を使用せずに、対応する表やビューをアクセスすることができます。

シノニム名が重複しない限り、表やビューに複数のシノニムを作成することができます。シノニムに表の完全修飾名を与え、所有者名を付けずにシノニム名で表を参照します。ユーザーがシノニムと同名の表を所有している場合、DBMasterは常に表の方を使用しシノニムを無視します。表やビューを削除すると、表やビューの全てのシノニムも自動的に削除されます。

システム表以外の任意の表のシノニムを削除することができます。システム表のシノニムは、DBMasterによって内部的に管理され削除することができません。

使用例

➲ **使用例　Employeesinfo**表に作成されたシノニム**Staff**を削除します、以下の構文を使用します：

```
DROP SYNONYM Staff;
```

或いは：

```
DROP SYNONYM IF EXISTS Staff
```

関連SQL

CREATE SYNONYM

DROP TABLE

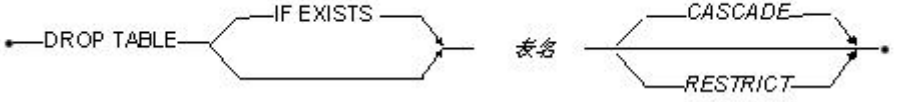
DROP VIEW

3.57 DROP TABLE

目的

データベースから表を削除します。

構文



IF EXISTS　　*表が存在しない場合エラーを起こさないこしないことを確保します*

表名	*データベースから削除した表の名*
CASCADE	*表に索引、外部キー、シノニム、ビューとトリガー 等のような従属するオブジェクトを削除移動します*
RESTRICT	*表に索引、外部キー、シノニム、ビューとトリガー 等のような従属するオブジェクトがない表を削除します*

説明

DROP TABLE文は表を削除します。このSQL文は、表所有者、DBA、SYSADMだけが実行することができます。

表を削除すると、表上の全ての索引と主キーも削除されます。表の主キーを参照する外部キーがあるときは、表を削除する前に、主キーを参照する全ての外部キーを削除しておきます。

CASCADE/RESTRICTのキーワードはオプションです。これらのキーワードは、削除されるテーブルで参照される関連したオブジェクトを削除するか、確認するかを指定します。**CASCADE**のキーワードが指定されると、テーブルに従属するオブジェクトはすべて削除されます。RESTRICTのキーワードが指定されると、従属するオブジェクトのないテーブルだけが削除できます。

使用例

➲ **使用例**　以下の文を使用して表**Employeesinfo**を削除します：

```
DROP TABLE Employeesinfo ;
```

或いは：

```
DROP TABLE IF EXISTS Employeesinfo
```

関連SQL

CREATE TABLE

ALTER TABLE ADD COLUMN

ALTER TABLE DROP COLUMN

ALTER TABLE MODIFY COLUMN

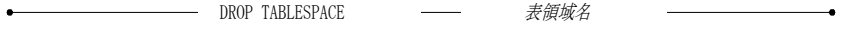
ALTER TABLE RENAME

3.58 DROP TABLESPACE

目的

データベースから表領域を削除します。

構文



表領域名　　削除する表領域の名前

説明

DROP TABLESPACE文は表領域を削除します。このSQL文は、DBAまたはSYSADMだけが実行することができます。

表領域を削除すると、表領域の全ての論理ファイルが自動的に削除されます。論理ファイルに対応する物理ファイルは、オペレーティング・システムコマンドを使い手動で削除してディスクを開放します。表領域を削除する前に、表領域内の全ての表を削除しておきます。

使用例

➲ **使用例**　表領域**ts_emp**を削除します；表領域を削除する前に、全ての表を削除しておきます：

```
DROP TABLESPACE ts_emp;
```

関連SQL

CREATE TABLESPACE

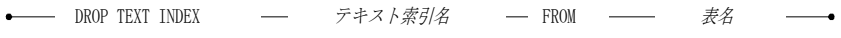
ALTER TABLESPACE

3.59 DROP TEXT INDEX

目的

表のテキスト索引を削除します。

構文



テキスト索引名	削除するテキスト索引の名前
表名	テキスト索引を削除する表の名前

説明

DROP TEXT INDEX文は、表カラム上のシグネ―チャもしくはIVFテキスト索引をデータベースから削除します。表所有者、DBA、SYSADM、表のINDEX権限をもつユーザーだけがこのSQLを実行することができます。

テキスト索引は、表のテキストカラムから語や句を含むテキスト行を高速にアクセスするための機構です。テキスト索引には、テキストカラムの全てのテキストの内部表現データがあります。データは符号化・構造化されており、直接表から検索するよりも高速に検索することができます。テキスト索引を作成しても、そのオペレーションはユーザーには見えません。DBMSが全文検索のパフォーマンスを上げるために使用します。

使用例

➲ **使用例　Employeesinfo**表のテキスト索引**TxtIdx**を削除します：

```
DROP TEXT INDEX TxtIdx FROM Employees info;
```

関連SQL

CREATE TEXT INDEX

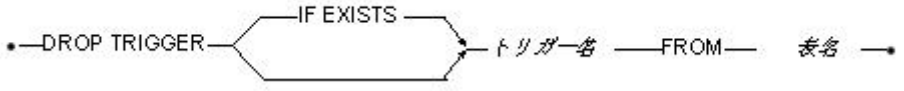
REBUILD TEXT INDEX

3.60 DROP TRIGGER

目的

表のトリガーを削除します。

構文



IF EXISTS	*トリガーが存在しない場合エラーを起こしないことを確保します*
トリガー名	削除するトリガーの名前
表名	トリガーを削除する表の名前

説明

DROP TRIGGER文はトリガーを削除します。このSQL文は、表所有者、DBA、SYSADMだけが実行することができます。

トリガーは、特定のイベントに応じて自動的に事前に定義されたSQL文を実行するデータベース・サーバーの機構です。トリガーは、通常は実行不可能な複雑かつ非類型的なデータベースオペレーションを実行可能にしま

す。トリガーはデータベース・サーバーの制御下にあり、イベントの如何に関わらず、データが整合して処理されることを保証します。トリガーオペレーションは、ユーザーには見えずに実行されます。トリガーは、ユーザーまたはアプリケーションプログラムがイベントを生成するごとに起動します。

トリガーは、トリガーが参照する表やカラムを削除する、カラム定義を変更する、BEFORE／AFTERキーワードを使用してカラムを追加すると不正になり使用できなくなります。BEFORE／AFTERキーワードを使用せずにカラムを追加したときは、トリガーは何の影響も受けません。不正なトリガーは、削除してデータベースから削除します。表に作成されたトリガーは、表が削除されると自動的に削除されます。

使用例

➲ **使用例**　以下の文を使用して**Employeesinfo**表のトリガー**Trig_emp**を削除します：

```
DROP TRIGGER Trig_emp FROM Employeesinfo;
```

或いは：

```
DROP TRIGGER IF EXISTS Trig_emp FROM Employeesinfo;
```

関連SQL

CREATE TRIGGER

ALTER TRIGGER ENABLE

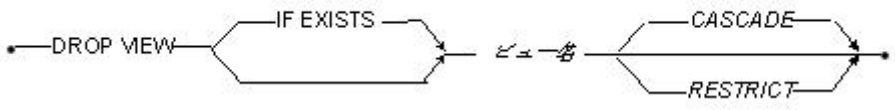
ALTER TRIGGER REPLACE

3.61 DROP VIEW

目的

データベースからビューを削除します。

構文



IF EXISTS ビューが存在しない場合エラーを起こさないことを確保します

ビュー名 データベースから削除するビューの名前

説明

DROP VIEW文はビューを削除します。このSQL文は、ビューの所有者、DBA、SYSADMだけが実行することができます。

ビューが削除されると、そのビューに基づくすべてのビューを無効にします。システムビューは削除できません。

CASCADE/RESTRICTのキーワードはオプションです。これらのキーワードは、削除されるビューで参照される関連したオブジェクトを削除するか、確認するかを指定します。**CASCADE**のキーワードが指定されると、ビューに従属するオブジェクトはすべて削除されます。**RESTRICT**のキーワードが指定されると、ビュー定義やシノニムで参照されるビューは削除されません。**RESTRICT**が指定されると、従属するオブジェクトがないビューだけが削除できます。

使用例

➲ **使用例** 以下の構文を使用してビュー**SalesStaff**を削除します：

```
DROP VIEW SalesStaff;
```

或いは：

```
DROP VIEW IF EXISTS SalesStaff;
```

➲ **使用例** 以下はどんなシノニムまたはビューが参照していても、**SalesStaff**という名前のビューを削除しません:

```
DROP VIEW SalesStaff RESTRICT
```

関連SQL

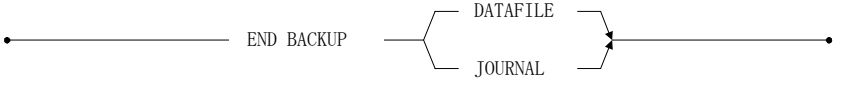
CREATE VIEW

3.62 END BACKUP

目的

オンライン・バックアップしたデータベースを通常の操作に戻します。

構文



説明

END BACKUP文は、オンライン・バックアップ中のデータベースのバックアップ状態を終了させます。このSQL文は、DBAまたはSYSADMだけが実行することができます。

オンライン完全バックアップを取るときは、NON-BACKUP、BACKUP-DATA、BACKUP-DATA-AND-BLOBモードでデータベースを起動しておきます。バックアップを開始するには、BEGIN BACKUPを実行します。オペレーティング・システムのコマンドまたはバックアップ・ユーティリティを使用して全てのデータファイルとBLOBファイルをバックアップしたら、END BACKUP DATAFILEを実行します。次にオペレーティング・システムのコマンドまたはバックアップ・ユーティリティを使用して全てのジャーナル・ファイルをバックしたら、END BACKUP JOURNALを実行し

てバックアップを完了し、データベースを通常の操作に戻します。オンライン完全バックアップを使用すると、END BACKUP DATAFILEを実行した時点から現在のアクティブ・ジャーナル・ファイルをコピーした時点までのデータベースにリストアすることができます。

```
BEGIN BACKUP; //手動に全てのデータファイルをコピーします
END BACKUP DATAFILE; //手動に全てのジャーナルファイルをコピーします
END BACKUP JOURNAL; //完全バックアップが終了した
```

オンライン差分バックアップを取るときは、NON-BACKUP、BACKUP-DATA或いはBACKUP-DATA-AND-BLOBモードでデータベースを起動します。差分バックアップは手動バックアップ方法を使用してバックアップを実行できません。差分バックアップは最新の完全バックアップのデータを基礎とします。つまり差分のベースです。差分バックアップは差分ベースを変更されたデータのみ含みます。

オンライン増分バックアップ或いは今のオンライン増分バックアップを取るときは、BACKUP-DATAかBACKUP-DATA-AND-BLOBモードでデータベースを起動しておきます。

現在までのオンライン増分バックアップを取るには、BACKUP-DATAまたはBACKUP-DATA-AND-BLOBモードでデータベースを起動しておきます。バックアップを開始するには、BEGIN INCREMENTAL BACKUP TO CURRENTを実行します。コピーを取る全てのジャーナル・ファイルと各ファイルのバックアップIDがリストされます。現在までのオンライン増分バックアップは、現在のアクティブ・ジャーナル・ファイルを含めて、前回までの完全バックアップ以降に使用された全てのジャーナル・ファイルをバックアップします。各ファイルのファイル名とバックアップIDを安全な場所に記録し、データベースをリストアするときに使用します。リストにあるジャーナル・ファイルのバックアップを取ったら、END BACKUP JOURNALを実行してバックアップを完了し、データベースを通常の操作に戻します。現在までのオンライン増分バックアップを使用すると、前回の完全バックアップのEND BACKUP DATAFILE実行時点から、現在のアクティブ・ジャーナル・ファイルがコピーされる時点までのデータベースにリストアすることができます。

オンライン・バックアップは、ABORT BACKUPを実行して随時中止することができます。詳細はABORT BACKUP文を参照してください。ABORT BACKUPを実行すると、そのバックアップ・ファイルを使用してデータベースをリストアすることはできません。バックアップ・ファイルは削除しておき、データベースをリストアするときに有効なバックアップ・ファイルと間違えないようにします。

使用例

➲ **使用例**　オンライン完全バックアップを取る手順を示します。**BEGIN BACKUP**を実行して完全バックアップを取ることを**DBMaster**に知らせます。全てのデータファイルと**BLOB**ファイルをバックアップ場所にコピーしたら、**END BACKUP DATAFILE**を実行します。全てのジャーナル・ファイルをバックアップ場所にコピーしてから**END BACKUP JOURNAL**を実行します。データベースは通常の操作に戻ります：

```
BEGIN BACKUP;
    Copy data and BLOB files to backup location using OS commands
    Change backup mode if desired
    Abort the backup if desired
END BACKUP DATAFILE;
    Copy Journal files to backup location using OS commands
    Change the backup mode if desired
    Abort the backup if desired
END BACKUP JOURNAL;
```

関連SQL

BEGIN BACKUP

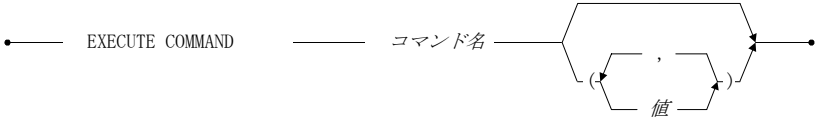
ABORT BACKUP

3.63 EXECUTE COMMAND

目的

ストアド・コマンドを実行します。

構文



コマンド名	実行するストアド・コマンドの名前
値	ストアド・コマンドのホスト変数の入力パラメータ

説明

EXECUTE COMMAND文は、ストアド・コマンドを実行します。ストアド・コマンドは、頻繁に使用されるSQLデータ操作文をコンパイルせずに素早く実行するのに使用します。このSQL文は、DBA、SYSADM、EXECUTE権限をもつユーザーだけが実行することができます。

ストアド・コマンドは、コンパイルされた実行形式のSQLデータ操作文で永続的にデータベースに格納されます。SQLデータ操作文をコンパイル・最適化せずに、繰り返しストアド・コマンドを実行することができます。ストアド・コマンドは、一つのSQLデータ操作文だけからなりプログラムロジックをもたない点を除けば、ストアド・プロシージャと類似しています。

ホスト変数は、ストアド・コマンドのSQLデータ操作文のカラム値の位置指定に使用されます。実際のカラム値は、コマンドの実行時に割当てま

す。ストアド・コマンドの中でホスト変数を指定するには、データやカラムの値を疑問符(?)で置き換えます。

ホスト変数をもつストアド・コマンドの実行時には、組み込み関数の結果値、NULLキーワード、DEFAULTキーワード等の定数、または他のホスト変数を使用します。組み込み関数は、RAND( )、PI( )、CURDATE( )、NOW( )のように引数の無いものだけを使用します。NULL値を与えるときは、ホスト変数のカラムがNULL値を受け入れるものでなければなりません。ストアド・コマンドの実行時に与えるパラメータの個数は、ストアド・コマンド定義時のホスト変数の個数と一致しなければなりません。

使用例

➲ **使用例1**　入力パラメータの無いストアド・コマンド**sc_select**.を実行します：

```
EXECUTE COMMAND sc_select.;
```

➲ **使用例2**　2つの入力パラメータがあるストアド・コマンド**sc_input**を実行します：

```
EXECUTE COMMAND sc_input (10002, 10006);
```

関連SQL

CREATE COMMAND

DROP COMMAND

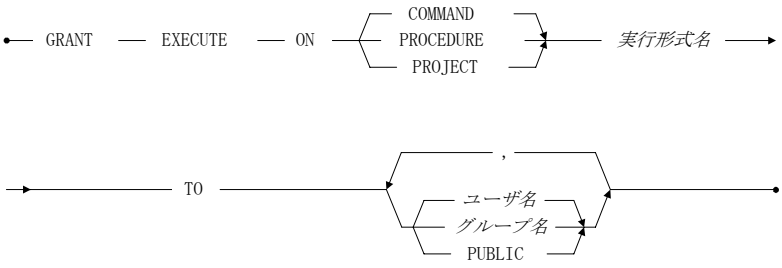
GRANT (実行権限)

3.64 GRANT（実行権限）

目的

データベースオブジェクトの実行権限をユーザーに与えます。

構文



実行形式名	実行権限を与える実行形式オブジェクトの名前
ユーザー名	実行権限が与えられるユーザーの名前
グループ名	実行権限が与えられるグループの名前

説明

GRANT文は、実行形式のデータベースオブジェクトの実行権限をユーザーに与えます。実行形式オブジェクトの所有者、DBA、SYSADMだけが実行することができます。

EXECUTE権限は、ユーザーが実行することができる実行形式データベースオブジェクトを制御します。DBMasterには3種類の実行形式オブジェクト、ストアド・コマンド、ストアド・プロシージャ、プロジェクトがあります。

COMMANDキーワードは、ストアド・コマンドのオブジェクトを指定します。ストアド・コマンドを構成するSQL文の実行に必要な全てのセキュリティ権限とオブジェクト権限をもち、実行権限をもつユーザーだけがストアド・コマンドを実行することができます。

PROCEDUREキーワードは、ストアド・プロシージャのオブジェクトを指定します。ストアド・プロシージャの実行権限だけが与えられます。

PROJECTキーワードは、複数のストアド・プロシージャを含むプロジェクトのオブジェクトを指定します。プロジェクトの実行権限は、自動的にプロジェクト内の全てのストアド・プロシージャの実行権限を与えます。

実行形式のデータベースオブジェクトの作成者がオブジェクトの所有者です。所有者とDBAおよびSYSADMは、自動的にオブジェクトの実行権限をもちます。実行権限を全てのユーザーに与えるにはPUBLICにします。実行形式のデータベースオブジェクトの実行権限が現在と将来の全てのユーザーに与えられます。

使用例

- **使用例1**　ストアド・コマンド**ListUserTables**の実行権限を**Vivian**に与えます：

```
GRANT EXECUTE ON COMMAND ListUserTables TO Vivian;
```

- **使用例2**　ストアド・プロシージャ**ShowUsers**の実行権限を**Jenny**と**John**、およびグループ**Managers**に与えます：

```
GRANT EXECUTE ON PROCEDURE ShowUsers TO Jenny, John, Managers;
```

- **使用例3**　プロジェクト**InternetFunc**にある全てのストアド・プロシージャの実行権限を**PUBLIC**キーワードを使用して全てのユーザーに与えます：

```
GRANT EXECUTE ON PROJECT InternetFunc TO PUBLIC;
```

関連SQL

CREATE GROUP

GRANT (セキュリティ権限)

GRANT (オブジェクト権限)

REVOKE (実行権限)

REVOKE (オブジェクト権限)

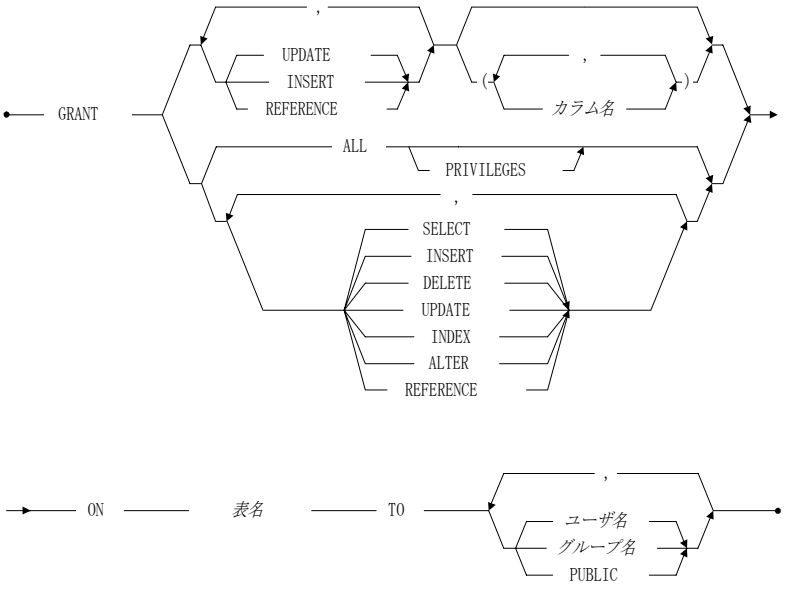
REVOKE (セキュリティ権限)

3.65 GRANT（オブジェクト権限）

目的

データベースオブジェクトの権限をユーザーに与えます。

構文



カラム名	オブジェクト権限を与えるカラムの名前
表名	オブジェクト権限を与える表の名前
ユーザー名	オブジェクト権限が与えられるユーザーの名前
グループ名	オブジェクト権限が与えられるグループの名前

説明

GRANT文は、データベースオブジェクトのアクセス権限を個々のユーザーに与えます。このSQL文は、オブジェクトの所有者、DBA、SYSADMだけが実行することができます。

オブジェクト権限は、ユーザーがアクセスすることができるデータベースオブジェクトと実行できるアクションを制御します。SELECT、INSERT、DELETE、UPDATE、INDEX、ALTER、REFERENCEのオブジェクト権限があります。キーワードALLとALL PRIVILEGESを使用して、全てのオブジェクト権限を同時に与えることができます。

- SELECT権限は、データベースオブジェクトのデータを検索するのに使用されます。権限はオブジェクト全体に適用されます。特定のカラムに限定することはできません。
- INSERT権限は、データベースオブジェクトにデータを挿入するのに使用されます。特定のカラムに権限を限定することもできます。
- DELETE権限は、データベースオブジェクトからデータを削除するのに使用されます。権限はオブジェクト全体に適用され、特定のカラムに限定することはできません。
- UPDATE権限は、データベースオブジェクトのデータを更新するのに使用されます。特定のカラムに権限を限定することもできます。
- INDEX権限は、データベースオブジェクトに索引を作成するのに使用されます。権限はオブジェクト全体に適用され、特定のカラムに限定することはできません。
- ALTER権限は、データベースオブジェクトのスキーマを変更するのに使用されます。権限はオブジェクト全体に適用され、特定のカラムに限定することはできません。

- REFERENCE権限は、データベースオブジェクトに外部キーのような参照制約を作成するのに使用されます。特定のカラムに権限を限定することもできます。

スキーマオブジェクトの作成ユーザーは、オブジェクト所有者になります。オブジェクト所有者、DBA、SYSADMは、自動的に全てのオブジェクト権限をもちます。システムカタログ表は、SYSTEMと呼ばれる仮想ユーザーが所有します。全てのユーザーは、システムカタログ表のSELECT権限だけをもちます。システムカタログ表の他のオブジェクト権限をユーザーに与えることはできません。

一つのGRANT文で特定カラムの権限とデータベースオブジェクト全体の権限を与えることはできません。この場合は、GRANT文を二度使用します。PUBLICに権限を与えることによって、全てのユーザーに権限を与えることができます。現在および将来の全てのユーザーにデータベースオブジェクトの権限が与えられます。

使用例

➲ **使用例1**　ユーザー**Vivian**に**Checks**表の**SELECT、INSERT、UPDATE**オブジェクト権限を与えます：

```
GRANT SELECT, INSERT, UPDATE ON Checks TO Vivian;
```

➲ **使用例2**　ユーザー**Jenny**に**Checks**表の**Amount**、**PayDate**カラムの**INSERT、UPDATE、REFERENCE**権限を与えます：

```
GRANT INSERT, UPDATE, REFERENCE (Amount, PayDate) ON Checks TO Jenny;
```

➲ **使用例3**　ユーザー**John**に**Checks**表の全てのオブジェクト権限を与えます：

```
GRANT ALL ON Checks TO John;
```

関連SQL

CREATE GROUP

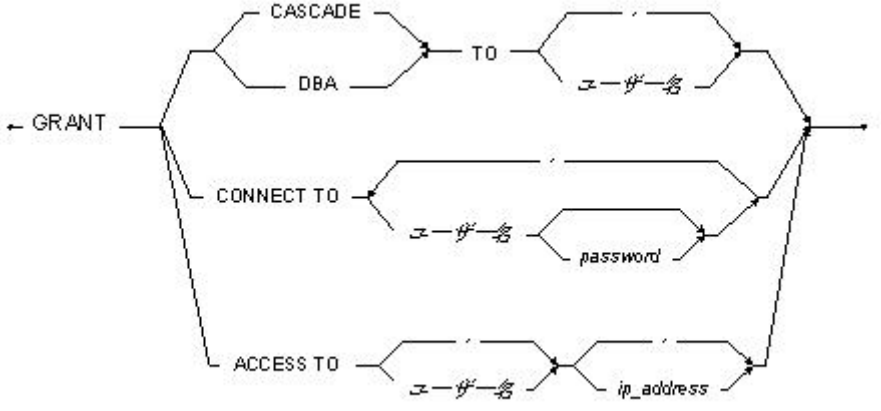
GRANT (実行権限)

3.66 GRANT（セキュリティ権限）

目的

データベースのセキュリティ権限をユーザーに与えます。

構文



ユーザー名	セキュリティ権限を与えるユーザーの名前
password	*データベース接続時のユーザーのパスワード*
ip_address	*セキュリティ権限を与えるユーザーのアドレス*

説明

GRANT文は、新規ユーザーを作成し既存ユーザーのセキュリティ権限を変更します。このSQL文は、SYSADMだけが実行することができます。データベースを作成すると、初期設定ユーザーSYSADMがパスワード無しで作成されます。データベース作成後、直ちにSYSADMのパスワードを変更して不正なアクセスを防止します。SYSADMは、他のユーザーにセキュリティ権限が与えられるまでは、唯一のデータベースユーザーになります。

SYSADMは、CONNECT、RESOURCE、DBA、ACCESSセキュリティ権限をユーザーに与えることができます。CONNECTセキュリティ権限を与えると、ユーザー名がデータベースに登録されます。SYSADMは、既存のユーザーに高位のセキュリティを与えることもできます。高位権限ユーザーは低位権限ユーザーの全ての権限を持ちます。他のユーザーにセキュリティ権限を与えることができるのは、SYSADMだけです。

CONNECTセキュリティ権限は、ユーザーがデータベースに接続するのに必要な権限です。CONNECTセキュリティ権限が与えられると、ユーザーとしてデータベースに登録されます。全てのユーザーは、他のセキュリティ権限が与えられる前に、CONNECTセキュリティ権限が与えられていなければなりません。CONNECTセキュリティ権限をもつユーザーは、一時表を作成したり、検索許可されたデータを問い合わせることができます。

RESOURCEセキュリティ権限は、表、ドメイン、索引を作成・変更・削除できるようにします。RESOURCE権限をもつユーザーは、作成したオブジェクトの所有者として、他のユーザーにオブジェクト権限を与えたり取り消すことができ、所有するオブジェクトのシノニムとビューを作成することができます。

DBA権限は、RESOURCE権限をもつ他に、表領域とファイルを作成することができます。DBA権限をもつユーザーは、システムスキーマオブジェクトを除いて、他のユーザーが所有するスキーマオブジェクトのオブジェクト権限を別のユーザーに与えたり取り消すこともできます。

ユーザー名の最大長は32字、パスワードの最大長は16字です。文字、数字、アンダースコア、記号$と#を使用することができます。1字目は数字以外にします。

ACCESS権限を使うと、特定のIPからデータベースに接続できます。これにより、データベースの保護と不正接続の防止が可能となります。IPは標準のインターネットプロトコル形式です。数字と‘*’アスタリスクだけが使用できます。

使用例

- **使用例1**　ユーザー**vivian**と**jenny**にパスワード無しの**CONNECT**権限を与えます：

```
GRANT CONNECT TO vivian, jenny;
```

- **使用例2**　ユーザー**vivian**と**jenny**にパスワード**shuka828**と**grala833**を付けて**CONNECT**権限を与えます：

```
GRANT CONNECT TO vivian shuka828, jenny grala833;
```

- **使用例3**　ユーザー**vivian**と**jenny**に**RESOURCE**権限を与えます：

```
GRANT RESOURCE TO vivian, jenny;
```

- **使用例4**　ユーザー**vivian**と**jenny**に**DBA**権限を与えます：

```
GRANT DBA TO vivian, jenny;
```

- **使用例5**　以下は、アドレス **192.4.55.3** および **219.3.44.*** と共に、**ACCESS** 権限をユーザー **Vivian** および **jenny** に与えます。

```
GRANT ACCESS TO vivian,jenny ‘192.4.55.3’,’219.3.44.*;
```

関連SQL

CONNECT

CREATE GROUP

GRANT (実行権限)

GRANT (オブジェクト権限)

REVOKE (実行権限)

REVOKE (オブジェクト権限)

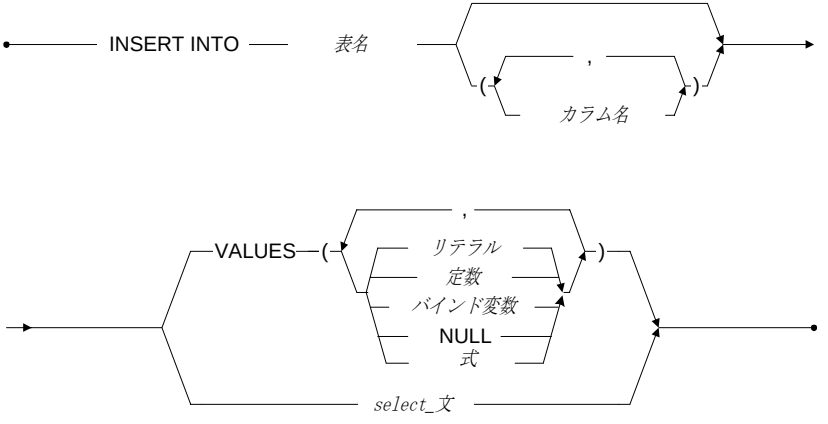
REVOKE (セキュリティ権限)

3.67 INSERT

目的

表に新しい行を挿入します。

構文



表名	新しい行を挿入する表の名前
カラム名	値を挿入するカラムの名前
リテラル	挿入するリテラル値
定数	挿入する定数
バインド変数	挿入するバインド変数の名前 (ODBC のみ)
式	挿入する式

説明

INSERT文は表に新しい行を挿入します。ただし、システムカタログ表には行を挿入することはできません。このSQL文は、表所有者、DBA、SYSADM、表または特定のカラムのINSERT権限をもつユーザーだけが実行することができます。

単一の行を挿入するときは、VALUESキーワードを使用して行の値を与えます。値は、リテラル値、定数、組み込み関数値、ODBC APIを使用したプログラムのバインド変数、式で与えます。SELECT文を使用して他の表から検索したデータ行の集まりを挿入することもできます。検索された行のカラムは、挿入表と互換のデータ型でなければなりません。

INSERT文のカラム名は、任意の順序で値を与えるカラムを指定します。カラム名リストを省略すると、表作成時の順に全カラムを指定したことになります。この場合、空のカラムにもカラム値を与えます。カラムのデータ型と一致しないデータが与えられたときは、適切なデータ型の値に変換されます。値が与えられないときは、カラムの初期設定値が使用されます。

親表にリンクする外部キーをもつ子表にデータを挿入するときは、参照整合性の規則が適用されます。子キーの値がNULLの場合を除いて、親キーに無い子キーの値を挿入することはできません。先に親キーの値の行を親表に挿入しておきます。

文字列に引用符(‘)を挿入するときは、二つの連続する引用符(‘’)を使用します。引用符を偶数個にしないと、DBMasterは文字列値を閉じてしまいます。初期設定値を挿入するには、値を空のままにするかDEFAULTキーワードを使用します。

使用例

➲ **使用例1　Employeesinfo**表に一行を挿入します：

```
INSERT INTO Employeesinfo VALUES (1234, ‘John’, ‘01/01/1998’, 2500);
```

➲ **使用例2**　**Employeesinfo** 表の**Emp_ID**、**FName**、**HireDate**カラムに値を挿入します：

```
INSERT INTO Employeesinfo (Emp_ID, FName, HireDate)
     VALUES (1234, 'John', '01/01/1998');
```

➲ **使用例3**　**Employeesinfo** 表の **Emp_ID**、 **FName**、 **HireDate** カラムに **TempStaff**表から**Emp_ID >10567**条件で検索した行の集まりを挿入します：

```
INSERT INTO Employeesinfo (Emp_ID, FName, HireDate)
     SELECT Emp_ID, FName, HireDate FROM TempStaff WHERE Emp_ID  > 10567;
```

➲ **使用例4**　引用符を含む文字列を**CHAR**カラムに挿入します。他のカラムは**DEFAULT**キーワードを使用して初期設定値にします：

```
INSERT INTO TB_TMP VALUES ('Joe''s Diner', DEFAULT, DEFAULT);
```

関連SQL

DELETE

LOCK TABLE

SELECT

UPDATE

3.68 KILL CONNECTION

目的

データベースにログオンしているユーザーの接続を切断します。

構文

接続_ID　　切断する接続番号

説明

KILL CONNECTION文は、データベースとのユーザー接続を切断します。DBAまたはSYSADMだけが実行することができます。

このSQL文は、ユーザーがロックしている全てのリソースを開放します。他のユーザーのプライオリティの高い操作で必要とするリソースを抱えているユーザーを切断する、あるいは、データベースを終了するときにログオフしていないユーザーを切断するのに使用します。

使用例

➲ **使用例**　接続_**ID**が**12345**のユーザー接続を切断します：

```
KILL CONNECTION 12345;
```

関連SQL

CONNECT

TERMINATE DATABASE

3.69 LOAD STATISTICS

目的

データベースの統計データをテキストファイルからロードします。

構文

ファイル名　　ロードする統計データをもつファイルの名前

説明

LOAD STATISTICS文は、データベースの統計データをもつテキストファイルから統計値をロードします。統計ファイルは、UNLOAD STATISTICS文を使用して作成します。ASCIIテキストエディタを使用して統計ファイルを編集し、テストあるいは他の目的で統計データを変更するすることができます。DBAまたはSYSADMだけが実行することができます。

使用例

➲ **使用例**　統計ファイル**stat.dat**をデータベースにロードします：

```
LOAD STATISTICS FROM stat.dat;
```

関連SQL

UNLOAD STATISTICS

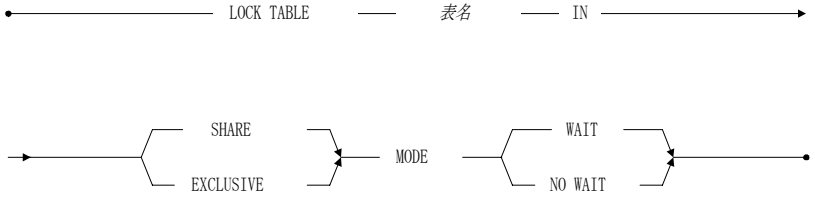
UPDATE STATISTICS

3.70 LOCK TABLE

目的

他のユーザーの表アクセスを制御します。

構文



表名　　ロック設定を変更する表の名前

説明

LOCK TABLE文は、他のユーザーの表アクセスを制御します。表所有者、DBA、SYSADM、SELECT権限をもつユーザー（SHAREモードにロック）、UPDATE／DELETE権限をもつユーザー（EXCLUSIVEモードにロック）だけがこのSQL文を実行することができます。

このSQL文は、表をSHAREまたはEXCLUSIVEモードにロックして他のユーザーの表アクセスを制御します。SHAREモードは、表読み込みは許しますが書き出しは禁止します。他のユーザーは、表行のINSERT、UPDATE、DELETEができなくなります。EXCLUSIVEモードは、読み込みと書き出しの両方を禁止します。他のユーザーは、表行のSELECT、INSERT、UPDATE、DELETEができなくなります。

LOCK TABLEは、操作中のデータベースのロック数を減らすのにも使用します。表の初期設定ロックがページまたは行のときに、多数の低レベルロックを避けるためにLOCK TABLEを使用して表レベルでロックします。DBMasterは、ロック数が多くなると自動的にロックレベルを上げるので、通常はLOCK TABLEを使用する必要はありません。

WAIT／NO WAITキーワードはオプションです。これらのキーワードは、表を直ちにロックすることができないときに、ロックできるまで待つかど

うかを指定します。NO WAITオプションは、ロックできるまで待たずにエラーメッセージを返します。ロックの待ち時間は、**dmconfig.ini**ファイルのDB_LTIMOキーワードの値で決められます。初期設定はWAITです。

使用例

➲ **使用例1　WAIT**オプションを付けて**Employeesinfo**表を**SHARE**モードにロックします：

```
LOCK TABLE Employeesinfo IN SHARE MODE WAIT;
```

➲ **使用例2　NO WAIT**オプションを付けて**Employeesinfo**表をEXCLUSIVEモードにロックします：

```
LOCK TABLE Employeesinfo IN EXCLUSIVE MODE NO WAIT;
```

関連SQL

CREATE TABLE

ALTER TABLE SET OPTIONS

DELETE

INSERT

SELECT

UPDATE

3.71 REBUILD COMMAND

目的

コマンドを再構築します。

構文

コマンド名　　*再構築するストアドコマンドの名*

説明

REBUILD COMMANDはストアドコマンドを再構築します、DBA、SYSADM権限をもつユーザーだけが実行することができます。

このコマンドはストアドコマンドを実行する場合の効率が低くなることを避けます。例えば、レコードがある表にストアドコマンドを作成すると、レコードを増加に伴って、ストアドコマンドを実行する効率は低くなります。

ユーザーはREBUILD COMMAND構文を使用してストアドコマンドを再構築できます、或いは状態の更新に伴ってデータベースは自動にストアドコマンドを再作成します。

使用例

➲ **使用例**　以下は**sc_select**というストアドコマンドを再構築します：

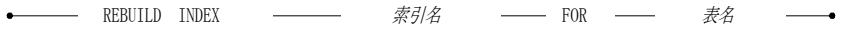
```
REBUILD INDEX NameIndex FOR Employeesinfo IN ts_new;
```

3.72 REBUILD INDEX

目的

表の索引を再構築します。

構文



索引名	再構築する索引の名前
表名	索引を再構築する表の名前

説明

REBUILD INDEX文は、表の既存の索引を再構築します。このSQL文は、表所有者、DBA、SYSADM、表のINDEX権限をもつユーザーだけが実行することができます。

索引は、キーカラム(複数)の値から特定の表行を素早くアクセスできるようにする機構です。索引にはキーカラムと同じデータがありますが、データは高速検索が可能なように構造化されソートされています。索引を作成しても、そのオペレーションはユーザーには見えません。DBMSは可能な限り索引を使用して問合せのパフォーマンスを良くします。

表の索引を再構築すると、高密度の断片化していない索引が作成され効率が良くなります。

使用例

➲ **使用例　Employeesinfo**表の索引**NameIndex**を再構築します：

```
REBUILD INDEX NameIndex FOR Employeesinfo;
```

関連SQL

CREATE INDEX

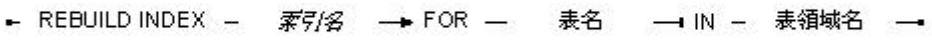
DROP INDEX

3.73 REBUILD INDEX IN ANOTHER TABLESPACE

目的

ほかの表領域に索引を再構築します。

構文



索引名　　*再構築する索引の名*

表名　　*索引を再構築する表の名*

表領域名　　*表領域の名*

説明

REBUILD INDEX IN ANOTHER TABLESPACEコマンドは別の表領域の表に索引を再構築して、本来の索引は自動に削除されます。表所有者、DBA、表のINDEXとALTER権限をもつユーザーだけがこのコマンドを実行することができます。

注　ユーザはTMPTABLESPACEに標準表の索引を再構造できません。

注　TMPTABLESPACEに一時表の索引だけを再構造できます。

注　表の索引をSYSTABLESPACEだけに再構造できます。

使用例

➲ **使用例**　以下の例は表領域**ts_new**に表**Employeesinfo**の索引**NameIndex**を再構造します、表**Employeesinfo**は表領域**ts_mode**にあります：

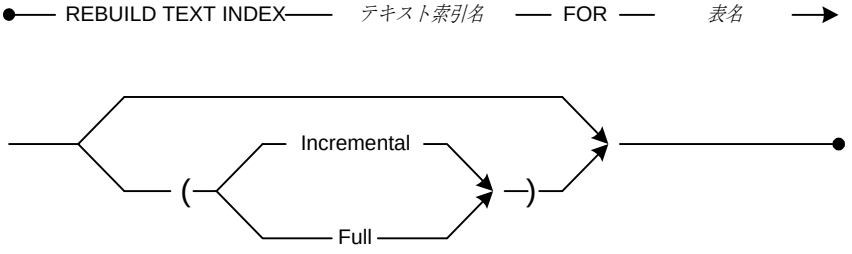
```
REBUILD INDEX NameIndex FOR Employeesinfo IN ts_new;
```

3.74 REBUILD TEXT INDEX

目的

新規データを含めて表のテキスト索引を再構築します。

構文



テキスト索引名	再構築するテキスト索引の名前
表名	テキスト索引を再構築する表の名前

説明

REBUILD TEXT INDEX文は、新規データが表のシグネ―チャもしくはIVFテキスト索引に含まれるように更新し再構築します。REBUILD TEXT INDEXは、表所有者、DBA、SYSADM、表のINDEX権限をもつユーザーだけが実行することができます。

テキスト索引は、テキストカラムから語や句を含むテキスト行を高速にアクセスする機構です。テキスト索引には、テキストカラム上の全てのテキストが符号化・構造化されており、表を直接検索するよりも高速に検索することができます。テキスト索引を作成しても、そのオペレーションはユーザーには見えません。DBMSが全文検索のパフォーマンスを上げるために使用します。

テキスト索引は、表にデータをロードしても更新されません。テキスト索引を作成する前に、全てのデータを表にロードしておきます。テキスト索引作成後に挿入された行のテキストは、全文検索の結果には返されません。これらの行を検索結果に含めるには、REBUILD TEXT INDEX文を使用してテキスト索引を再構築します。

INCREMENTALオプションは、テキスト索引を作成した後に表に入力されたテキストを追加します。つまりテキストを全文検索で利用できるようにします。FULLオプションは、テキスト索引全体を削除し、新しい全文検索に基づいて索引を再編成します。初期設定は、INCREMENTALです。

使用例

➲ **使用例　Employeesinfo**表のテキスト索引**TxtIdx**を再構築します：

```
REBUILD TEXT INDEX TxtIdx FOR Employeesinfo;
```

関連SQL

CREATE TEXT INDEX

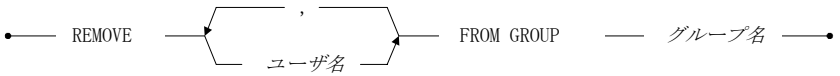
DROP TEXT INDEX

3.75 REMOVE FROM GROUP

目的

グループからユーザーを削除します。

構文



ユーザー名	グループから削除するユーザーの名前
グループ名	ユーザーを削除するグループの名前

説明

REMOVE FROM GROUP文は、既存のグループからユーザーを削除します。ユーザーは、グループに与えられているオブジェクト権限を全て失いますが、直接ユーザーに与えられた権限は残ります。このSQL文は、SYSADMまたはDBAユーザーだけが実行することができます

グループは、多数のユーザーをもつデータベースのオブジェクト権限の管理を容易にします。グループは、同じ権限を必要するユーザーをまとめるのに使用します。グループに与えられたオブジェクト権限は、自動的にグループのユーザー全員に与えられます。

グループに加えられたメンバーは、直接ユーザーに与えられたオブジェクト権限の他に、グループに与えられたオブジェクト権限を取得します。

使用例

➲ **使用例1**　グループ**SalesStaff**からユーザー**Vivian**を削除します：

```
REMOVE Vivian FROM GROUP SalesStaff;
```

➲ **使用例2**　グループ**SalesStaff**からグループ**NYSalesStaff**を削除します：

```
REMOVE NYSalesStaff FROM GROUP SalesStaff;
```

関連SQL

CREATE GROUP

ADD TO GROUP

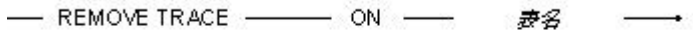
DROP GROUP

3.76 REMOVE TRACE

目的

詳細なOLD/NEWデータをログするシングル表からトレースを削除します。

構文



表名　既存なシングル表の名

説明

実際に、表の所有者、DBAまたは SYSADMのみはこのADD TRACEコメントを実行することができます。

3.77 RESUME SCHEDULE

目的

一時停止した非同期表レプリケーションのスケジュールを再開します。

構文

ターゲット・データベース名　レプリケーション・スケジュールを再開するターゲット・データベースの名前

説明

RESUME SCHEDULE文は、一時停止した非同期表レプリケーションのスケジュールを再開します。このSQL文は、ソース表の所有者、DBA、SYSADMだけが実行することができます。

使用例

➲ **使用例**　ターゲット・データベース**DivOneDb**のレプリケーション・スケジュールを再開します：

```
RESUME SCHEDULE FOR REPLICATION TO DivOneDb;
```

関連SQL

ALTER SCHEDULE

CREATE REPLICATION

CREATE SCHEDULE

DROP REPLICATION

DROP SCHEDULE

SUSPEND SCHEDULE

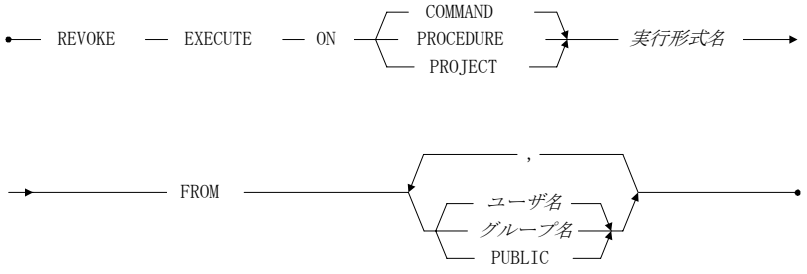
SYNCHRONIZE SCHEDULE

3.78 REVOKE （実行権限）

目的

ユーザーのデータベースオブジェクトの実行権限を取り消します。

構文



実行形式名	実行権限を取り消す実行オブジェクトの名前
ユーザー名	実行権限を取り消すユーザーの名前
グループ名	実行権限を取り消すグループの名前

説明

REVOKE文は、ユーザーまたはグループに与えられたデータベースオブジェクトの実行権限を取り消します。このSQL文は、オブジェクト所有者、DBA、SYSADMだけが実行することができます。

EXECUTE権限は、ユーザーが実行できる実行形式データベースオブジェクトを制御します。DBMasterには3種類の実行形式オブジェクト、ストアド・コマンド、ストアド・プロシージャ、プロジェクトがあります。

COMMANDキーワードは、ストアド・コマンドの実行権限取り消しを指定します。ストアド・コマンドを構成するSQL文の実行に必要な全てのセキ

ュリティ権限とオブジェクト権限をもち、実行権限をもつユーザーだけがストアド・コマンドを実行することができます。

PROCEDUREキーワードは、ストアド・プロシージャの実行権限取り消しを指定します。ストアド・プロシージャの実行権限だけが取り消されます。

PROJECTキーワードは、複数のストアド・プロシージャを含むプロジェクトの実行権限取り消しを指定します。プロジェクトの実行権限を取り消すと、自動的にプロジェクト内の全てのストアド・プロシージャの実行権限が取り消されます。

実行形式データベースオブジェクトの所有者、DBA、SYSADMは、自動的に実行権限をもちます。PUBLICの実行権限を取り消すと、全てのユーザーの実行権限が一斉に取り消されます。全てのユーザーは、実行形式名で指定するデータベースオブジェクトの実行権限を失います。

使用例

➲ **使用例1**　ストアド・コマンド**ListUserTables**の実行権限をユーザー**Vivian**から取り消します：

```
REVOKE EXECUTE ON COMMAND ListUserTables FROM Vivian;
```

➲ **使用例2**　ストアド・プロシージャ**ShowUsers**の実行権限をユーザー**Jenny**と**John**およびグループ**Managers**から取り消します：

```
REVOKE EXECUTE ON PROCEDURE ShowUsers FROM Jenny, John, Managers;
```

➲ **使用例3**　**PUBLIC**キーワードを使用して、**InternetFunc**プロジェクトに格納されている全てのストアド・プロシージャの実行権限を現在および将来の全てのユーザーから取り消します：

```
REVOKE EXECUTE ON PROJECT InternetFunc FROM PUBLIC;
```

関連SQL

DROP GROUP

GRANT (実行権限)

GRANT (セキュリティ権限)

GRANT (オブジェクト権限)

REVOKE (オブジェクト権限)

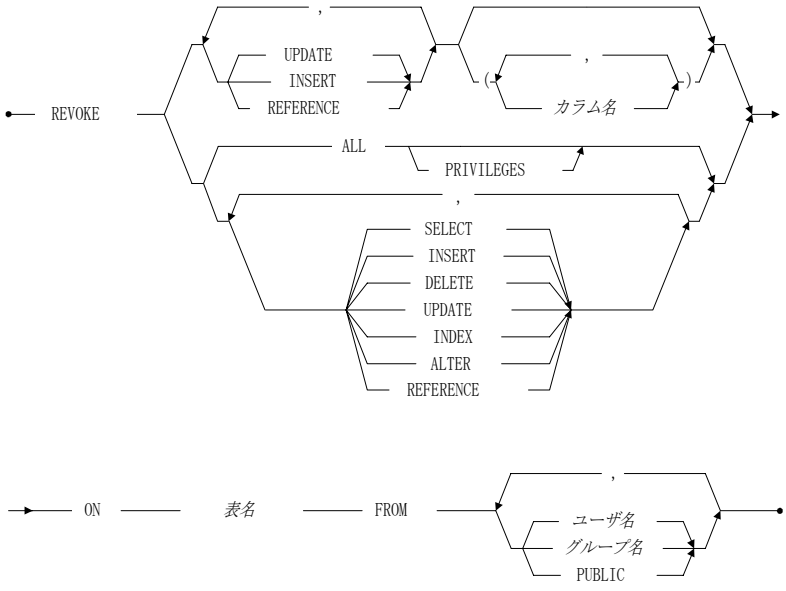
REVOKE (セキュリティ権限)

3.79 REVOKE （オブジェクト権限）

目的

ユーザーのオブジェクト権限を取り消します。

構文



カラム名	オブジェクト権限を取り消すカラムの名前
表名	オブジェクト権限を取り消す表の名前
ユーザー名	オブジェクト権限が取り消されるユーザーの名前
グループ名	オブジェクト権限が取り消されるグループの名前

説明

REVOKE文は、ユーザーまたはグループに与えられたデータベースオブジェクトのアクセス権限を取り消します。このSQL文は、オブジェクト所有者、DBA、SYSADMだけが実行することができます。

オブジェクト権限は、ユーザーがアクセスできるデータベースオブジェクトと実行できるアクションを制御します。7つのオブジェクト権限、SELECT、INSERT、DELETE、UPDATE、INDEX、ALTER、REFERENCEがあります。キーワードALLとALL PRIVILEGESを使用して、全てのオブジェクト権限を同時に取り消すことができます。

- SELECT権限は、データベースオブジェクトからデータを検索することを許可します。権限はオブジェクト全体に適用され、特定のカラムに限定することはできません。
- INSERT権限は、データベースオブジェクトに新規データを挿入することを許可します。特定のカラムに権限を限定することもできます。
- DELETE権限は、データベースオブジェクトからデータを削除することを許可します。権限はオブジェクト全体に適用され、特定のカラムに限定することはできません。
- UPDATE権限は、データベースオブジェクトのデータを更新することを許可します。特定のカラムに権限を限定することもできます。
- INDEX権限は、データベースオブジェクトに索引を作成することを許可します。権限はオブジェクト全体に適用され、特定のカラムに限定することはできません。
- ALTER権限は、データベースオブジェクトのスキーマを変更することを許可します。権限はオブジェクト全体に適用され、特定のカラムに限定することはできません。
- REFERENCE権限は、データベースオブジェクトに外部キーの参照制約を作成することを許可します。特定のカラムに権限を限定することもできます。

システムカタログ表の所有者は、SYSTEMと呼ばれる仮想的なユーザーです。全てのユーザー(SYSADMも)は、システムカタログ表のSELECT権限だけをもちます。システムカタログ表のオブジェクト権限を取り消すことはできません。

特定のカラムとオブジェクト全体に与えられた権限を取り消すときは、特定のカラムのREVOKE文と表全体のREVOKE 文の二つに分けて使用します。PUBLICの権限を取り消すことによって、全てのユーザーのオブジェクト権限を取り消すことができます。現在および将来の全てのユーザーは、データベースオブジェクトの権限を失います。

使用例

➲ **使用例1**　ユーザー**Vivian**に与えられた表**Checks**の**SELECT、INSERT、UPDATE**オブジェクト権限を取り消します：

```
REVOKE SELECT, INSERT, UPDATE ON Checks FROM Vivian;
```

➲ **使用例2**　ユーザー**Jenny**に与えられた表**Checks**のカラム**Amount、PayDate**の**INSERT、UPDATE、REFERENCE**オブジェクト権限を取り消します：

```
REVOKE INSERT, UPDATE, REFERENCE (Amount, PayDate) ON Checks FROM Jenny;
```

➲ **使用例3**　ユーザー**John**に与えられた表**Checks**の全ての権限を取り消します：

```
REVOKE ALL ON Checks FROM John;
```

関連SQL

DROP GROUP

GRANT (実行権限)

GRANT (オブジェクト権限)

GRANT (セキュリティ権限)

REVOKE (実行権限)

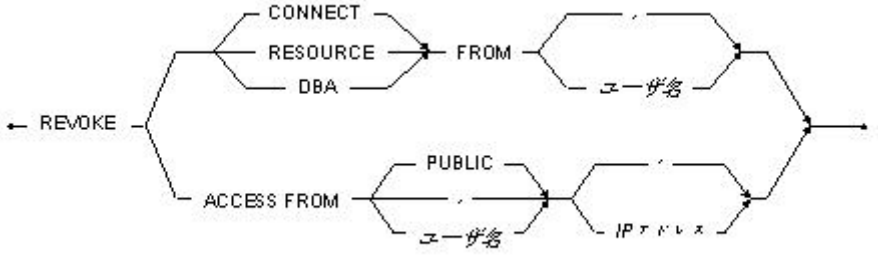
REVOKE (セキュリティ権限)

3.80 REVOKE （セキュリティ権限）

目的

ユーザーに与えられたデータベースのセキュリティ権限を取り消します。

構文



ユーザー名 セキュリティ権限が取り消されるユーザーの名前

IPアドレス **セキュリティ権限を取り消すユーザーのアドレス**

説明

REVOKE文は、データベースのユーザーを削除したり、ユーザーのセキュリティ権限を変更したりします。SYSADMだけが実行することができます。

SYSADMは、ユーザーのDBA、RESOURCE、CONNECT、ACCESS権限を取り消すことができます。CONNECT権限を取り消すと、他のセキュリティ権限も取り消されます。ユーザー名はデータベースから削除され、ユーザーはデータベースに接続できなくなります。CONNECT以外のセキュリティ権限を取り消しても、その他のセキュリティ権限は取り消されません。

DBA権限は、RESOURCE権限をもつだけでなく、表領域とファイルを作成することもできます。DBA権限をもつユーザーは、システムスキーマオブ

ジェクトを除いて、他のユーザーが所有するスキーマオブジェクトのオブジェクト権限を別のユーザーに与えたり取り消すことができます。

RESOURCE権限は、表、ドメイン、索引を作成・変更・削除できるようにします。RESOURCE権限をもつユーザーは、作成したオブジェクトの所有者として、他のユーザーにオブジェクト権限を与えたり取り消すことができ、所有するオブジェクトのシノニムとビューを作成することができます。

CONNECT権限は、ユーザーがデータベースに接続するのに必要な権限です。CONNECT権限が与えられると、ユーザー名がデータベースに登録されます。CONNECT権限は、他のセキュリティ権限が与えられる前に、全てのユーザーに与えられていなければなりません。CONNECT権限をもつユーザーは、一時表を作成したり、検索許可が与えられているデータをデータベースに問い合わせることができます。

ACCESS権限を使うと、特定のIPからデータベースに接続できます。これにより、データベースの保護と不正接続の防止が可能となります。IPは標準のインターネットプロトコル形式です。数字と ‘*’ だけが使用できます。

使用例

➲ **使用例1**　ユーザー**vivian**と**jenny**の**DBA**権限を取り消します：

```
REVOKE DBA FROM vivian, jenny;
```

➲ **使用例2**　ユーザー**vivian**と**jenny**の**RESOURCE**権限を取り消します：

```
REVOKE RESOURCE FROM vivian, jenny;
```

➲ **使用例3**　ユーザー**vivian**と**jenny**の**CONNECT**権限を取り消します。全ての権限が取り消されると共にデータベースユーザーから削除されます：

```
REVOKE CONNECT FROM vivian, jenny;
```

➲ **使用例4**　以下は、アドレス **192.55.3.4** および **219.5.3.*** と共に、**Vivian** および **jenny** という名前のユーザーの**ACCESS**権限を取り消します：

```
REVOKE ACCESS FROM Vivian,jenny '192.55.3.4','219.5.3.*'
```

関連SQL

DROP GROUP

GRANT (実行権限)

GRANT (オブジェクト権限)

GRANT (セキュリティ権限)

REVOKE (実行権限)

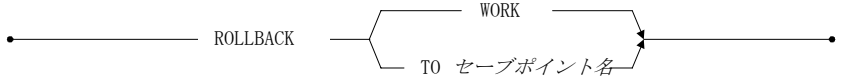
REVOKE (オブジェクト権限)

3.81 ROLLBACK

目的

トランザクションまたはトランザクションの一部をロールバックします。

構文



セーブポイント名	ロールバックするセーブポイントの名前

説明

ROLLBACK文は、トランザクションの開始時点またはセーブポイントを設定した時点にトランザクションをロールバックします。CONNECT以上の権限をもつユーザーは誰でも実行することができます。

ROLLBACK文は、トランザクション内のSQL文が変更したデータを全て元に戻します。ROLLBACKすると、トランザクションの全てのロックが解除

されます。データベースが AUTOCOMMITモードで作動しているときは、このSQL文は機能しません。

ROLLBACK文は、トランザクション内のSQL文が変更したデータの一部を元に戻すこともできます。セーブポイント後に実行したSQL文だけをロールバックし、セーブポイント前の全てのSQL文はロールバックしません。トランザクションはそのまま残り、ロックは解除されません。

使用例

➲ **使用例1** トランザクションをロールバックし、アボートと同じ結果になります。トランザクションが掛けたロックは全て解除されます：

```
ROLLBACK WORK;
```

➲ **使用例2** セーブポイント**SavePoint1**の後に実行した全てのSQL文をロールバックしますが、セーブポイントの前に実行したSQL文はそのままにします；トランザクションはそのまま残り、ロックは解除されません：

```
ROLLBACK TO SavePoint1;
```

関連SQL

BEGIN WORK

COMMIT WORK

SAVEPOINT

3.82 SAVEPOINT

目的

セーブポイントを設定します。

構文

セーブポイント名　　セーブポイントに付ける名前

説明

SAVEPOINT文は、トランザクションに名前付きのセーブポイントを設定します。ユーザーは誰でも実行することができます。

SAVEPOINT文は、トランザクション内の一部のSQL文だけをロールバックするROLLBACK文と共に使用されます。ROLLBACK文でセーブポイント名を指定すると、設定したセーブポイント以降のSQL文をロールバックします。トランザクションはそのまま残り、ロックは解除されません。

存在しないセーブポイント名を指定すると、トランザクション全体がロールバックされてエラーが返されます。トランザクションはアボートされ、トランザクションが掛けた全てのロックは解除されます。トランザクション内で同じセーブポイント名を二度付けると、最初のセーブポイントはキャンセルされ、第二のセーブポイントに名前が付けられます。

使用例

➲ **使用例**　トランザクションにセーブポイント名**SavePoint1**のセーブポイントを設定します：

```
SAVEPOINT SavePoint1;
```

関連SQL

BEGIN WORK

COMMIT WORK

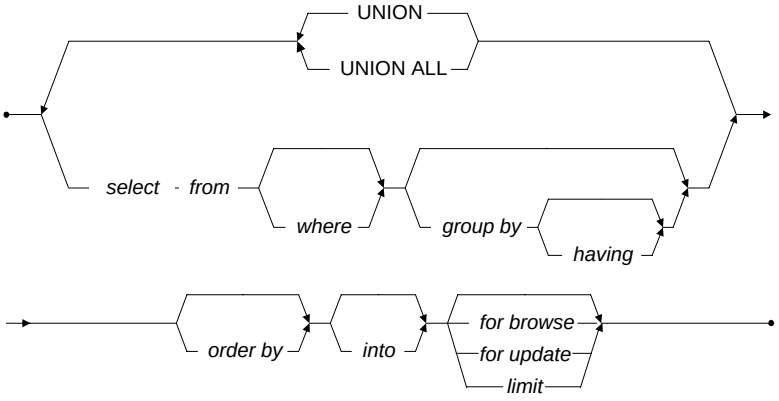
ROLLBACK

3.83 SELECT

目的

データベースのデータを検索します。

構文



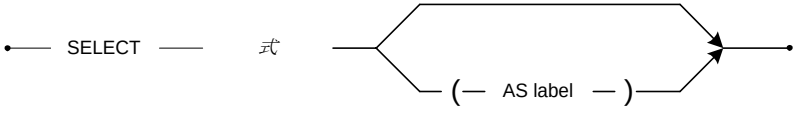
select	検索カラムをリストするSELECT句
from	検索カラムの表をリストするFROM句
where	検索すべき行の条件を指定するWHERE句
group by	集計グループを指定するGROUP BY句
having	グループのフィルタ条件を指定するHAVING句
order by	ソート順を指定するORDER BY句
into	検索結果を格納する表を指定するINTO句
for browse	ブラウズモード検索を指定するFOR BROWSE句
for update	更新モード検索を指定するFOR UPDATE句
limit	結果レコードの数を指定するLIMIT句

説明

SELECT文は、データを探索・検索・表示します。表所有者、DBA、SYSADM、SELECT権限をもつユーザーだけがSELECT文を実行することができます。

SELECT文の結果は、指定された条件を満たす行の集まりです。これを結果セットといいます。SELECT文には、問い合わせる表やビュー、結果セットのデータが満たすべき条件、結果セットのデータの出力順序などを指定します。UNIONを使用して複数のSELECT文の結果セットを合併することができます。

FROM句を使用しないSELECT文



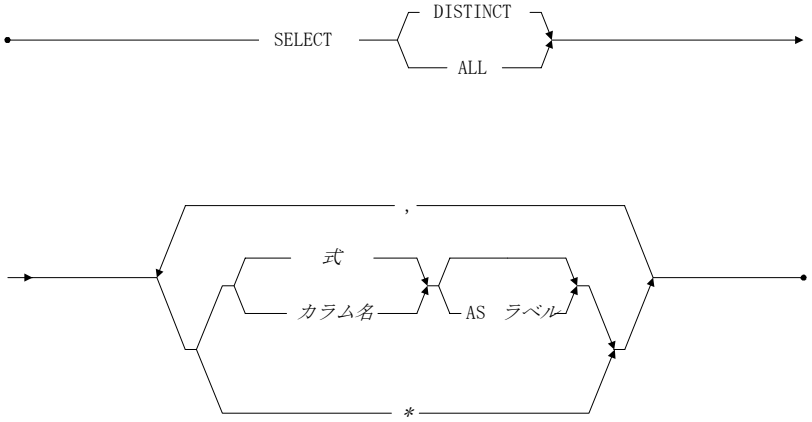
FROM句を使用しないSELECT文を用いて、UDFあるいは式の結果を取得します。問合せの中にFROM句を使う必要はありません。つまり、カラム名や表名を指定することはできません。

FROMを用いないSELECT文に次の句を用いることはできません。WHERE、GROUP BY、HAVING、ORDER BY、DISTINCT、UNION。

➲ **使用例：**

```
SELECT abs(100), cos(100.0);
```

SELECT句



式	結果セットに入れる値をもつ式
カラム名	結果セットに入れる検索カラムの名前
ラベル	検索カラム名とは別の結果セットカラムの名前

SELECT句は、SELECTキーワードとデータベースオブジェクトまたは結果セットに入れる値の式のリストから成ります。ALLとDISTINCTキーワードは、重複行を結果セットに含めるかどうかを指定します。どちらも指定しないときは、初期設定で重複行を含む全ての行が返されます。

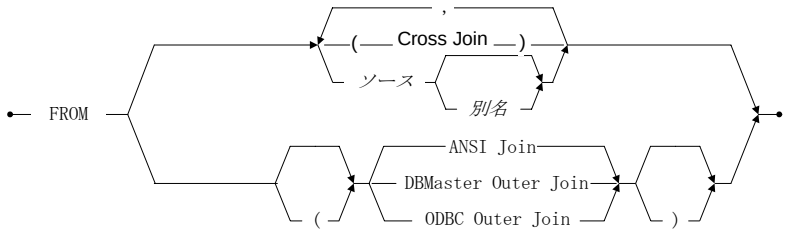
SELECT項目は、カラム名、式、定数、アスタリスク(*)で指定します。アスタリスクは検索表の全カラムを表します。カラム名とアスタリスクの前に検索表名を付けることができます。

SELECT項目に指定する式は、カラム、定数、関数、集計関数の任意の基本タイプの式です。定数を指定すると、全ての行の項目に同じ値が返されます。集計関数は一連の行に対して一つの値を返します。集計関数は、通常、GROUP BY句と共に使用します。

OIDをカラム名に使用すると、表の隠しカラムであるOIDの値が返されます。OIDは、データベースにある表の各行を一意に識別する値です。OIDの値がシーケンスになっている保証はありません。

ASキーワードを使用すると、結果セットのカラムや式に一時的な表示ラベルを付けることができます。

FROM句



ソース	検索表の名前
別名	ソースに付ける別の名前（他の句で使用）

FROM句は、検索表やビューのソースをリストします。ソースはカラム名の出所表を明らかにします。ソースは、表名かビュー名か問い合わせ結果かシノニム名のいずれかです。ソースには、単一ソースとOUTERソースがあります。OUTERソースは、OUTERキーワードの後に単一ソースを一つ以上続けて指定します。

表に別名を付け、別名で表を参照して文を読み易くすることができます。自己結合(self-join)の場合は、別名が必須になります。

➲ **使用例:**

次の問い合わせは、t2からc1カラムからの最大値に相当する値を選び、c2からの値によってそれらをグループ化します。最後に、結果セットは別前t3を与えられます。

SELECT * FROM (select max(c1) FROM t2 GROUP BY c2) AS t3 (c1);

外結合のキーワードOUTER、LEFT OUTER JOIN、LEFT JOINを使用して、複数の表を外結合することができます。SELECT文には、複数の外結合キーワードを入れることができます。外結合キーワードの前にある全てのソースは主ソース、外結合キーワードの後にある全てのソースは従ソースでなければなりません。全ての外結合表をFROM句で指定し、外結合条件をWHERE句で指定します。WHERE句の結合条件は、全て外結合条件として取り扱われます。他の条件は、外結合条件を評価する前に評価されます。

DBMasterは、ON句で外結合条件を指定するANSIとODBCの外結合構文もサポートします。WHERE句の条件は、外結合条件の後に評価されます。

CROSS JOINは、2つの表のクロス・プロダクトを指定します。WHERE句が古いタイプ（非SQL-92-型ジョイン）で指定されていない場合、同じ行が返ります。結果は、ユーザーがFROMの表リストに','を指定した場合と同じようになります。

➲ 使用例1

```
select * from t1 cross join t2 cross join t3 where t1.c1 = t2.c1 and t2.c2 =
t3.c3;
```

結果は以下の問合せと同じようになります。

➲ 結果

```
ex: select * from t1,t2,t3 where t1.c1 = t2.c1 and t2.c2 = t3.c3;
```

DBMaster 3.5以降のバージョンでは、問合せに使用する検索タイプと索引を指定することができます。更に、データベース統計を最新に更新していなくても、問合せを最適化し、自動的に最も効率的な検索タイプを決定することができます。

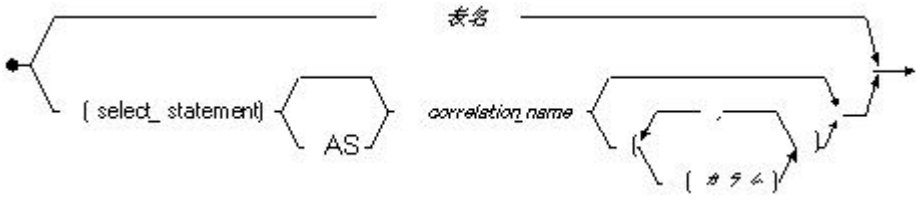
➲ 使用例2　索引検索を指定する構文を示します：

```
表名 (INDEX [=] 索引名 [ASC|DESC])
```

値0で表検索を強制し、値1で主キー索引検索を強制することもできます。

SOURCE 副句

FROM句の中で使用されるSOURCE 副句は問い合わせからセットされた表名あるいは結果のいずれかかもしれません。問い合わせからセットされた結果を使用するためには、以下の図の中で提供されるシンタックスを使用してください。



Correlation_name　　副クエリーの結果セットを表します。

強制インデックススキャン

下記の構文にてインデックスの強制スキャンを実行します：

```
tablename (INDEX [=] idxname [ASC|DESC])
```

値が0の場合、強制テーブルスキャンを、1の場合、主キー・インデックススキャンを実行します。

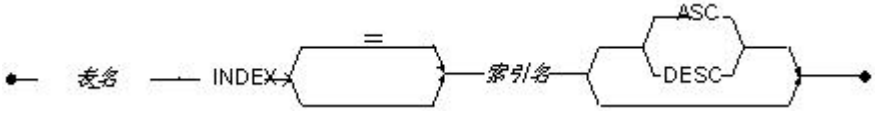


図3-92 強制インデックススキャン文字列

➲ **使用例1**　テーブルスキャンが値**0**を指定するよう̄に強制する方法：

```
SELECT * FROM tb_tmp (INDEX=0)
```

➲ **使用例2**　プライマリキーのインデックススキャンが値1を指定するように強制する方法:

```
SELECT * FROM tb_tmp (INDEX=1)
```

➲ **使用例3**　インデックス **idx1** のインデックススキャンを強制する方法:

```
SELECT * FROM tb_tmp (INDEX idx1)
```

➲ **使用例4**　クエリオプティマイザがテーブル t1 で使用するスキャンのタイプを決定するのを可能にしますが、テーブル **t2** の **idx1** インデックスでインデックススキャンを強制します:

```
SELECT * FROM t1, t2 (INDEX idx1)
```

強制インデックススキャンと"エイリアス"

テーブルのエイリアスを強制インデックススキャンに使用可能:

```
tablename (INDEX [=] idxname) aliasname
```

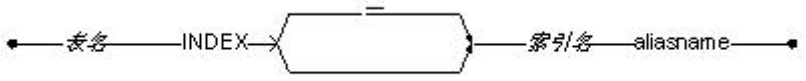


図3-93　インデックススキャン強制および'エイリアス'を使用した構文

➲ **使用例1**　**idx1** インデックスのインデックススキャンを強制し、テーブルにエイリアスを提供する方法:

```
SELECT * FROM t1 (INDEX idx1) a, t1 b WHERE a.c1 = b.c1
```

強制インデックススキャンと"シノニム"

シノニムを使用した強制インデックススキャン:

```
synonymname (INDEX [=] idxname)
```

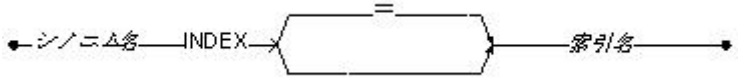


図3-94　インデックススキャン強制および'シノニム'を使用した構文

➲ **使用例1**　シノニムを使用してインデックス**idx1**の強制スキャンを行う方法:

```
SELECT * FROM s1 (INDEX idx1)
```

強制インデックススキャンと“ビュー”

ビュー作成時に強制インデックススキャン実行:

```
viewname (INDEX [=] idxname)
```

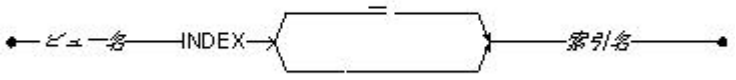


図3-95 強制インデックススキャンと‘ビュー’構文

➲ **使用例1**　ビュー v1 作成時にインデックス**idx1**の強制インデックススキャンを実行する方法:

```
CREATE VIEW v1 as SELECT * FROM t1 (INDEX idx1)
```

ビューSELECT時に強制インデックススキャンは実行できません。

➲ **使用例2**　エラーが返答される誤った使用方法:

```
SELECT * FROM v1 (INDEX idx1)
```

強制テキストインデックススキャン

強制テキストインデックススキャン構文:

```
tablename (TEXT INDEX [=] idxname)
```

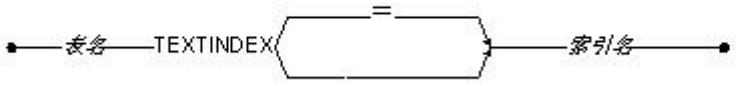


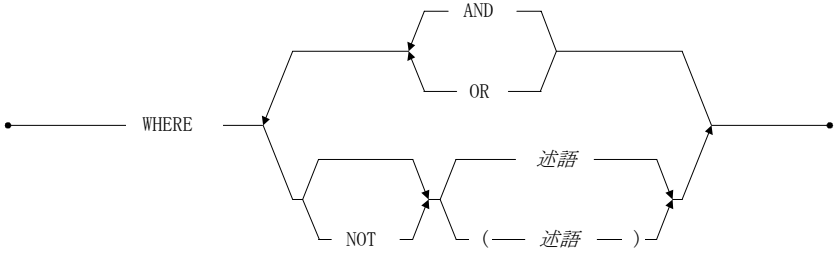
図3-96　強制テキストインデックススキャン構文

➲ **使用例1**

インデックスtidx1に強制テキストインデックススキャンを実行する構文:

```
SELECT * FROM t1 (TEXT INDEX tidx1)
```

WHERE句



WHERE句は、検索データの検索条件および結合条件を指定します。検索条件を満たす行だけが結果セットに入れられます。WHERE句の中で副問合せのSELECT文を使用する方法については、「副問合せ」の節を参照してください。

引用文字列では、ワイルドカードとしてパーセント(%)とアンダースコア(_)を使用します。パーセントは任意の0個以上の文字とマッチし、アンダースコアは任意の1文字とマッチします。*ESCAPE* はオプションです。ワイルドカードとして解釈しないパーセントとアンダースコアを引用文字列に含めるためのエスケープ文字を定義します。引用文字列に引用符を含めるには、二つの連続する引用符を使用します。

WHERE句で使用する述語には、以下の単純な比較演算があります：

- 関係演算子 — 関係演算子には >、>=、<=、<、=、<> があります。演算子の両辺の値が演算子の関係を満たすときに真になります。
- BETWEEN — x BETWEEN y AND z の形式で比較します。BETWEEN 条件は、x の値が y と z の値の範囲内にあるときに真になります。
- IN — x IN (y, z, ...) の形式で比較します。IN 条件は、x の値が (y, z, ...)リストのどれかであるときに真になります。
- IS NULL — x IS NULL の形式で比較します。IS NULL 条件は、x の値が NULL 値のときに真になります。
- IS NOT NULL — x IS NOT NULL の形式で比較します。IS NOT NULL 条件は、x の値が NULL 値以外のときに真になります。
- LIKE — x LIKE ‘y’ ESCAPE ‘z’ の形式で比較します。LIKE 条件は、文字列 x が引用文字列 ‘y’ の判定基準 (大文字と小文字を区別する)を満たすときに真になります。
- MATCH — x NOT CASE MATCH ‘y’ の形式で比較します。MATCH 条件は、x の文字列全体が引用文字列 ‘y’ にマッチするときに真になります。NOT キーワードは比較結果を否定し、CASEキーワードは大文字と小文字を区別します。日本語の場合は平仮名と片仮名を区別しますが、dmconfigファイルのキーワードDB_LCODE=2をセットする必要があります。いずれもオプションです。
- CONTAIN — x NOT CASE CONTAIN ‘y’ の形式で比較します。CONTAIN 条件は、x の任意の部分文字列が引用文字列 ‘y’ とマッチするときに真になります。NOTキーワードは比較結果を否定し、CASE キーワードは大文字と小文字を区別します。どちらもオプションです。
- CONTAINS – containsオペレーターの条件は、連結カラムからの連結したストリングが、ストリングのパターンと一致する時満たされます。

➲ **使用例:** 使用可能なシンタックス: **[NOT] CONTAINS (column || column [|| column]…, 'string pattern'[, option string])**次の**select文は、c1**および**c4**の両方がストリグ'**Mail Server**'を含んでいるc4からレコードを選ぶでしょう。オプション**CASE**は探索をケース英大小文字識別にします。

```
Select c4 from mcol where contains (c1 || c4, 'Mail Server', 'CASE');
```

CAST

あるデータ型から別のデータ型への変換を行います。有効な変換は以下の表のとおりです。表で示されるデータ型は、**X**（横）から**Y**（縦）に変換されます。

Numeric、Character、Date/Timeデータ型には、複数のデータ型が含まれます。Numericデータ型には、integer (int、serial)、smallint、float、double、decimalが含まれます。Characterデータ型には、charとvarcharが含まれます。Date/Timeデータ型には、date、time、timestampが含まれます。

xy	Int (serial)	Small-int	decimal	double	float	(Var) char	(var) binary	date	time	Time-stamp	file	blob	clob
Int(serial)	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
smallint	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
decimal	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
double	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
float	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
(Var)char	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N
(var)binary	N	N	N	N	N	Y	N	N	N	N	N	N	N
date	N	N	N	N	N	Y	N	Y	N	Y	N	N	N
time	N	N	N	N	N	Y	N	N	Y	N	N	N	N
Time-stamp	N	N	N	N	N	Y	N	Y	Y	Y	N	N	N
file	N	N	N	N	N	Y	Y	N	N	N	Y	N	N
blob	N	N	N	N	N	Y	Y	N	N	N	N	Y	Y
clob	N	N	N	N	N	Y	Y	N	N	N	N	Y	Y

➲ **使用例1　WHERE**句に**CAST( )**を使う：

```
SELECT * FROM t1 WHERE CAST(c1 AS CHAR(20)) like '2001%';
```

➲ **使用例2**　式の中に**CAST( )**を使う：

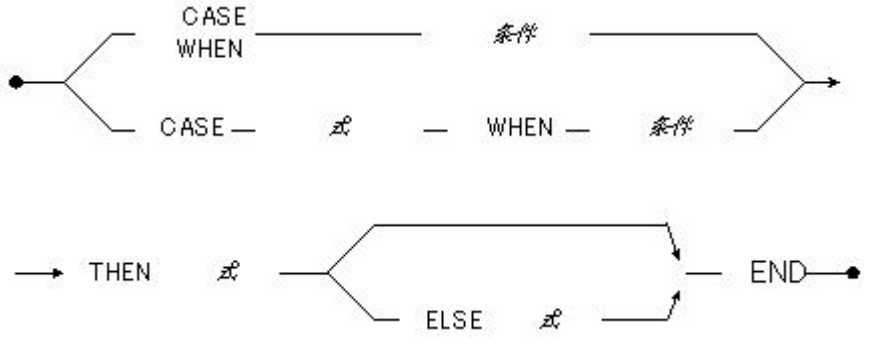
```
SELECT CAST(c1+c2 as CHAR(10)) FROM t1;
```

➲ **使用例3**　入れ子**CAST ( )**文を使う：

```
SELECT CAST(CAST(123 as CHAR(10)) || CAST(45 as CHAR(10)) as INT) FROM t1;
```

CASE

CASEは、SQL99関数です。



➲ **使用例1**

CASE WHEN p1 THEN v1 ELSE CASE WHEN p2 THEN v2 ELSE… ELSE vn END…ENDは、条件p1が真の場合v1になり、条件p2が真の場合v2になり、条件pnが真の場合vnになります。この文は、次のように実行します：

```
Select case when c1=3 then c2 else case when c1=5 then c3 else c4 end end
from t1;
```

➲ **使用例2**

CASE c1 WHEN d1 THEN v1 ELSE CASE c1 WHEN d2 THEN v2 ELSE…ELSE vn END…ENDは、c1=d1の場合v1になり、c1=d2の場合v2になり、cn=dnの場合vnになります。この文は、次のように実行します：

```
Select case c1 when 3 then c2 else case c1 when 5 then c3 else c4 end end
from t1;
```

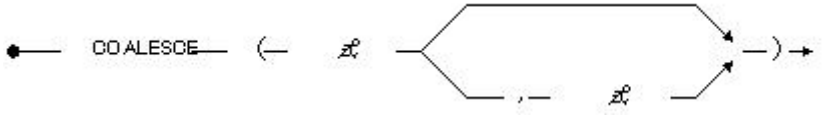
➲ **使用例3**

CASE WHEN p1 THEN v1 WHEN p2 THEN v2 WHEN…ELSE vn ENDは、p1が真の場合v1になり、p2が真の場合v2になり、pnが真の場合vnになります。この文は、次のように実行します：

```
Select case when c1=3 then c2 when c1=5 then c3 else c4 end from t1;
```

COALESCE

COALESCEは、SQL99関数です。COALESCE (v1, v2, v2,….vn)は、v1がNULLでなければv1、v2がNULLでなければv2、vnがNULLでなければvnになります。



➲ **使用例1**

```
SELECT COALESCE(c1, 7) FROM t1;
```

➲ **使用例2**

```
SELECT COALESCE(c1, c2, c3, 7) FROM t1;
```

NULLIF

NULLIFは、SQL99関数です。NULLIF(v1, v2)は、v1=v2の場合はNULL、さもなくばv1になります。

➲ **使用例1**

```
SELECT NULLIF(c1, 7) FROM t1;
```

➲ **使用例2**

```
SELECT NULLIF(t1.c1, t2.c1) FROM t1, t2;
```

IFNULL

IFNULL は ODBC 関数です。IFNULL (v1, v2) は coalesce(v1,v2) と同じで、“if v1 is not null, then v1 else v2” と同じです。

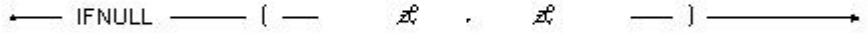


図3-102 IFNULL 文字列

➲ **使用例1**

```
Select ifnull(c1, 7) from t1;
```

➲ **使用例 2**

```
Select ifnull(t1.c1, t2.c1) from t1, t2;
```

複合条件

複合条件は、論理演算子AND、OR、NOTで単純述語を組み合わせて作成します。キーワードANDは、両方とも真でなければならない二つの条件を組み合わせます。キーワードORは、どちらか一方(あるいは両方)が真でなければならない二つの条件を組み合わせます。キーワードNOTは、条件が偽の行を検索します。

複合条件の**WHERE** 句を例示します。

➲ **使用例1：**

```
SELECT * from Custerm WHERE City NOT IN (‘LA’, ‘NY’) AND Age > 40;
```

➲ **使用例2：**

```
SELECT * From Orders WHERE Price > 10,000 OR Ship_Date = TODAY;
```

結合条件

結合条件は、二つの表のカラムを比較する関係演算です。

使用例：**Orders.CusNum = Customer.CusNum**

二つの表のカラムの結合条件をもつWHERE句は、関係付けられた二つの表を結合します。結合条件を満たす各々表行の対を一つの行にした一時合成表が結合結果になります。表結合には2表結合、マルチ表結合、自己結合、外結合、内部結合があります。

ON <検索条件>

ON <検索条件> は、ジョインが基にする条件を指定します。条件にはどのような述語も指定できます。一般的にはカラムと比較演算子を使用します。

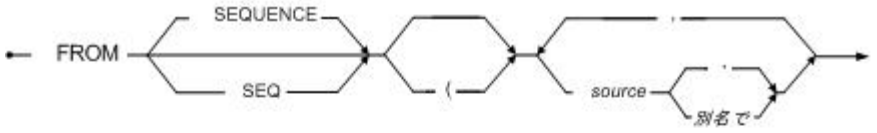
➲ **使用例**

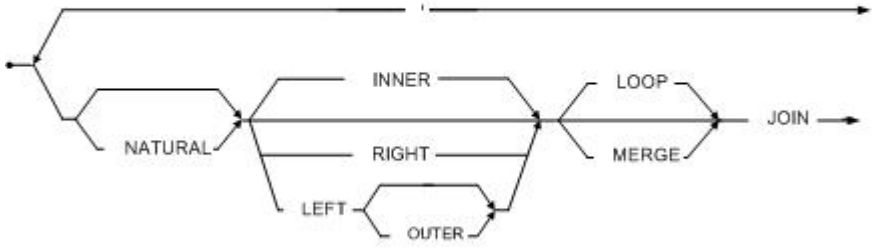
```
SELECT ProductID, Suppliers.SupplierID
FROM Suppliers JOIN Products
ON (Suppliers.SupplierID = Products.SupplierID)
```

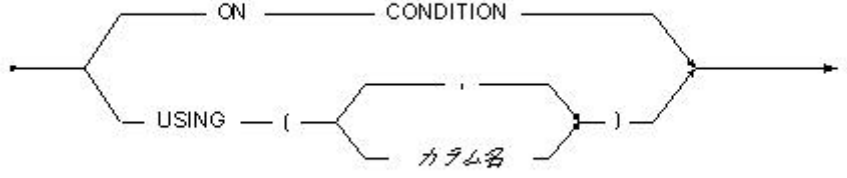
ANSI外部ジョイン

外部ジョインは、1対の表毎の外部ジョイン条件を持つ複数の表の結合です。外部ジョイン条件は、二つの表にあるカラムを比較する関係演算です。左側の全ての表のレコードが戻り、外部ジョイン条件がFALSEの場合、右側の表の結果はNULLになります。

以下のグラフは ANSI JOIN とオプティマイザヒント文字列を表示します:







SEQUENCE、**SEQ**、**LOOP**、**MERGE**キーワードはオプティマイザヒントとして使用されますが、ANSI 文字列ではありません。オプティマイザは指定されたキーワードが結合実行で使用される場合、実行計画を選択します。何も影響がない場合、オプティマイザはエラーメッセージを返しません。

SEQUENCE/SEQのキーワードが指定されると、コマンドのテーブル結合順序といった結合シーケンスが強制されます。結合テーブル実行シーケンスはオプティマイザでは変更されません。このキーワードは外部結合で使用される場合は何の影響も及ぼしません。

➲ **使用例**

```
select * from seq t1 inner join t2 on t1.c1=t2.c1 inner join t3 on t1.c2=t3.c2;
```

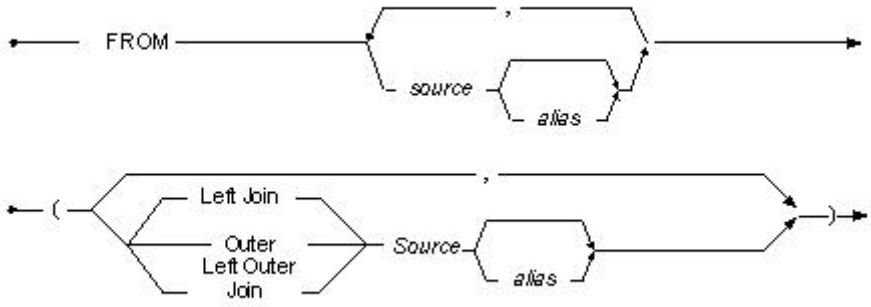
LOOP/MERGEのキーワードは内部および外部結合の結合実行方法を指定します。結合テーブルの結合実行順序は結合実行方法の指定時に変更されません。LOOP のキーワードが指定されると、オプティマイザは内部または外部結合にネステッド結合を使用します。MERGE のキーワードが指定されると、オプティマイザは同等結合による内部および外部結合にマージ結合を使用します。

➲ **使用例**

```
select * from t1 inner merge join t2 on t1.c1=t2.c1;
```

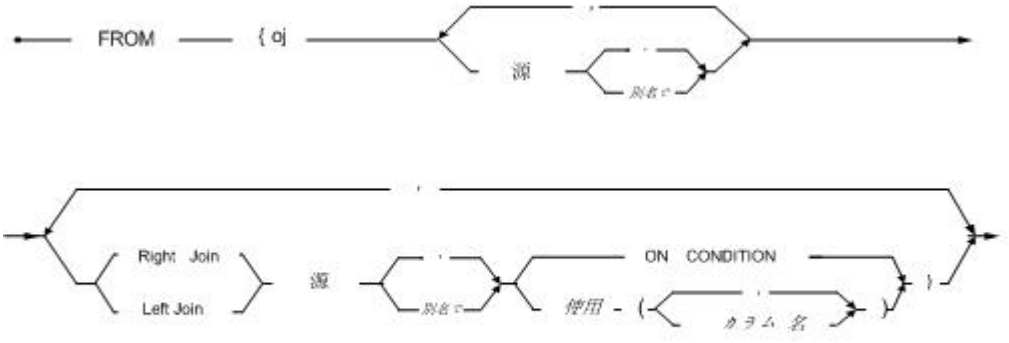
DBMaster外部ジョイン

以下の構文は、古いDBMasterの構文です。ANSI外部ジョイン構文との違いは、外部ジョイン要素がDBMasterのオプティマイザによって決定されることです。RIGHT-JOINは、以下の構文では使用することができません。ANSI外部ジョイン構文と次の構文を合わせて使うことができません。



ODBC外部ジョイン

ODBC外部ジョインは、ANSI外部ジョインと同じ構文を使います。



自己結合

自分自身の表と表結合します。**FROM**句で表名を二度リストし、各々に別名を付けます。**WHERE**句では、別名を使用して二つの表を参照します。**Employeesinfo**表の**Manager_ID**カラムには、従業員のマネージャの**Emp_Id**があるとします。

➲ **使用例　Employeesinfo**表を自己結合して、従業員名とマネージャ名を全てリストします：

```
SELECT e .FName AS Emp, m. Fname AS Manager
                                    FROM Employeesinfo e, Employeesinfo m
                                   WHERE e.Manager_Id = m.Emp_Id;
```

右結合

右結合は、結果セットに含まれるジョイン条件に合わない右側の表にある全ての行を指定します。他の表に対応する出力カラムはNULLに、更に内部結合によって戻される全行にセットされます。

➲ **使用例**

```
select * from t1 right join t2 on t1.c1 = t2.c1;
```

内部結合

内部結合は、合致する全ての対の行が戻ることを意味します。両方の表から合致しない行を破棄します。このジョイン型は、JOINキーワードが問合せの中に使われている場合のみ、初期設定になります。

➲ **使用例1**

```
select * from t1 inner join t2 on t1.c1 = t2.c1;
```

➲ **使用例2**

```
select * from t1 join t2 on t1.c1 = t2.c1;
```

➲ **使用例3**

```
select * from t1, t2 where t1.c1 = t2.c1;
```

NATURAL JOIN

NATURALキーワードがjoinタイプの前に指定されると、結合条件や結合カラムリストを指定するのにON条件やUSINGカラムリストは使用できません。NATURAL JOINは結合テーブルの共通カラム名で同等結合を実行しま

す。NATURAL JOINの結果はUSINGカラムリストのすべての共通カラム名を指定した場合と同じです。“select *” の射影リストは結合テーブルのカラムの残りに先行する結合カラムです。

➲ **使用例1**

```
select * from t1 natural inner join t2;
```

➲ **使用例2**

```
select * from t1 natural left join t2;
```

ON条件

ON条件は結合テーブルの結合条件を指定します。

➲ **使用例1**

```
select * from t1 inner join t2 on t1.c1 = t2.c1;
```

➲ **使用例2**

```
select * from t1 left join t2 on t1.c1 = t2.c1;
```

USINGカラムリスト

USINGのカラムリストは結合テーブルの結合カラムリストを指定するのに使用されます。USINGが指定されると、USINGカラムリストで指定されたすべてのカラム名は結合テーブルに存在し、比較可能となります。結果はON節のカラムで同等結合を指定する場合と同じです。“select *” の射影リストは結合テーブルのカラムの残りに先行する結合カラムです。

➲ **使用例1**

```
SELECT * FROM t1 INNER JOIN t2 USING (c1, c2);
```

➲ **使用例2**

```
select * from t1 left join t2 using (c1) left join t3 using (c1)
```

2表結合

2 表結合は、二つの表を結合条件を用いて結合します。

➲ **使用例　Dept_id**を使用して**Emp_Name**と**Dept_Name**を2表結合します：

```
SELECT FName, Dept_Name FROM Employeesinfo, Department
                          WHERE Employeesinfo.Dept_ID = Department.Dept_Id;
```

マルチ表結合

マルチ表結合は、表対毎の結合条件を用いて二つ以上の表を結合します。表対毎の結合条件は、二つの表上のカラムを比較する関係演算です。

➲ **使用例　Engineering**部門の**employeesinfo**が参加している全てのプロジェクトを 3 表結合によって検索します：

```
SELECT Dept_Name, Proj_Name FROM Department d, Project p, Employeesinfo e
                            WHERE d.Dept_id = e.Dept_Id AND
                                  p.Emp_Id = e.Emp_Id AND
                                 Dept_Name = 'Engineering';
```

強制ループ結合 (ネステッド結合)

2つのテーブルの間でネステッド結合を強制するのに使用される一般的な構文:

```
tablename { INNER | OUTER } LOOP JOIN tablename
```

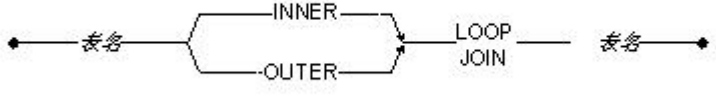


図3-107 強制ループ結合の文字列

このタイプの強制結合には、INNER JOINまたはOUTER JOINを使用してください。

➲ **使用例1**

```
SELECT * FROM t1 INNER LOOP JOIN t2 ON t1.c1=t2.c1;
```

➲ **使用例2**

```
SELECT * FROM t1 OUTER LOOP JOIN t2 ON t1.c1=t2.c1;
```

強制マージ結合

2つのテーブルの間でマージ結合を強制するのに使用される一般的な構文:

```
tablename { INNER | OUTER } MERGE JOIN tablename
```

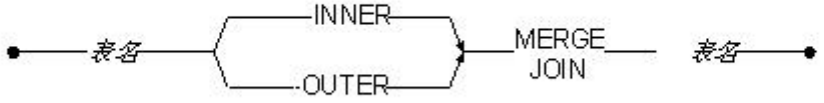


図3-108 強制マージ結合の構文

結合がマージ結合を使用できない場合、マージ結合の強制は無効になりますが、エラーメッセージは返されません。

➲ **使用例1**

```
SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.c1=t2.c1;
```

➲ **使用例2**

```
SELECT * FROM t1 OUTER MERGE JOIN t2 ON t1.c1=t2.c1;
```

強制結合シーケンス

すべてのテーブルの結合シーケンスを強制するので、結合シーケンスはスワップできません。強制結合シーケンス構文です:

```
SELECT ..... FROM [SEQUENCE | SEQ] tablename_list;
```

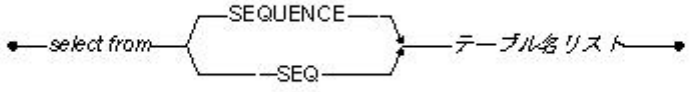


図3-109 強制結合シーケンスの構文

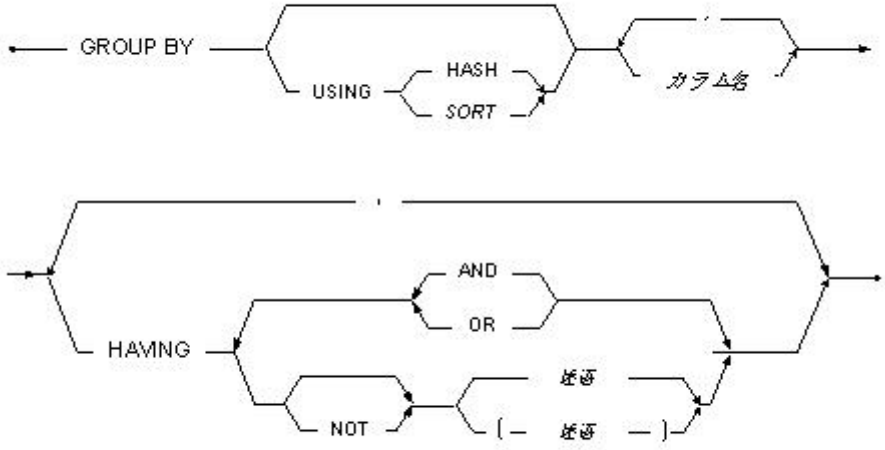
➲ **使用例1**

```
select * from sequence t1, t2, t3 where t1.c1=t2.c1 and t2.c2=t3.c2;
```

➲ **使用例2**

```
select * from seq t1 inner join t2 on t1.c1=t2.c1 inner join t3 on
t1.c2=t3.c2;
```

GROUP BY句



GROUP BY句は、グループの集計データを作成するときに使用します。**GROUP BY**で指定したカラムの値が同じ行をグループにして集計された行が作成されます。集計したカラムは、カラム名または表示ラベルで識別します。

➲ **使用例**　**Dept_Id**毎の従業員の**AVG(salary)**を集計します。次に、全従業員の**AVG(salary)**を取得するために**ID1**に集計します：

```
SELECT Dept_Id, AVG(Salary) FROM Employeesinfo GROUP BY Dept_Id;

SELECT Dept_Id AS ID1, AVG(Salary) FROM Employeesinfo GROUP BY ID1;
```

GROUP BY句はSELECTカラムリストに制限を加えます。GROUP BY句をもつSELECTの項目は以下のいずれかでなければなりません：

- グループに含まれる行を集計して単一の値を生成する集計関数
- GROUP BY句でリストされているグループカラム
- 定数

- これらを組み合わせた式.

実用的には、GROUP BY問合せには、常にグループカラムと集計関数が含まれます。GROUP BYカラムの値がNULLの行は、全て一つのグループに集められます。

USING HASH/SORT節はオプティマイザヒント構文として使用されます。USING HASHが指定されると、オプティマイザはGROUP BYの実行にハッシュメソッドを選択します。USING HASHの指定時にGROUP BYのグループが多すぎると、オプティマイザはハッシュメソッドを選択しません。USING SORTが指定されると、GROUP BY節と同じカラムを備えたインデックスがある場合、またはGROUP BY実行時にGROUP BYカラムによってソートする実行計画を選択される場合、オプティマイザはインデックススキャンを使用しようとします。

➲ **使用例1**

以下はSELECTを使って各従業員のDept_Id と AVG(salary) を検索してから、グループ全体の平均給与を得られるように従業員の AVG(salary) を ID 1 追加します。

```
SELECT Dept_Id, AVG(Salary) FROM Employeesinfo
                          GROUP BY Dept_Id;
SELECT Dept_Id AS ID1, AVG(Salary) FROM Employeesinfo
                          GROUP BY ID1;
```

➲ **使用例2**

以下はSELECTを使って、HASH メソッドにより Employeesinfo テーブルの各部門のDept_Id と AVG(salary) を検索します。

```
SELECT Dept_Id, AVG(Salary) FROM Employeesinfo
                          GROUP BY Dept_Id USING HASH;
```

強制 Group by メソッド

結合シーケンスの強制に使用される一般の構文:

```
GROUP BY column_name_list [USING SORT | USING HASH] having .....
```

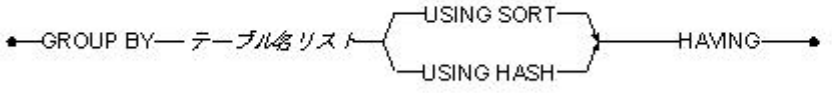


図3-111 強制 Group by メソッド構文

➲ **使用例1**

```
select c1,c2,count(*) from tb_test group by c1,c2 using hash;
```

➲ **使用例2**

```
select c1,c2,count( * )  from tb_test group by c1,c2 using sort having
sum(c3)>0;
```

HAVING句

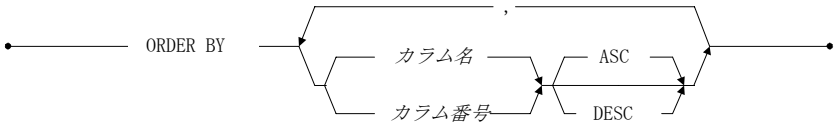
HAVING句は、グループを選択するか排除するときに使用します。

➲ **使用例**　売上げ合計が百万ドルを超える部門について、部門名と平均売上高をリストします：

```
SELECT Dept_Name, AVG(Amount) FROM Sales
                          GROUP BY Dept_Name
                         HAVING SUM(Amount) > 1000000;
```

HAVING句の中で副問合せを使用することができます。詳細については、「副問合せ」のセクションを参照してください。

ORDER BY 句



カラム名	問合せ結果をソートするSELECTリスト内のカラム名または表示ラベル
カラム番号	SELECTリスト内のカラムまたは式の位置を表す番号

ORDER BY句は、問合せ結果の行をカラム(複数)でソートします。ORDER BY句の無い問合せ結果は、特定の順序に並んではいません。

ASC／DESCキーワードは、ソートが昇順／降順のいずれかを指定します。初期設定は昇順です。NULL値は、非NULL値よりも大きいソート順になります。ASCキーワードでソート順を指定すると、非NULL値の後にNULL値が続きます。

➲ **使用例1**　名前の昇順、年齢の降順に検索結果をソートします：

```
SELECT Name, Address, Age FROM Customer
                        ORDER BY Name, Age DESC;
```

➲ **使用例2**　**ORDER BY**句の中でカラム番号と表示ラベルを使用します：

```
SELECT Dept_Id, Salary + Bounce AS Total_Com, FName
                                FROM Employeesinfo
                             ORDER BY 1, Total_Com;
```

UNIONオペレータ

UNIONオペレータは、複数の問合せ結果を一つに合併します。合併できる結果セットには以下の制限があります：

- 問合せ結果は、同じカラム数でなければなりません。
- 問合せ結果の対応項目は、同じカラム名である必要はありませんが、互換データ型をもたなければなりません。最初の問合せ結果のカラム名が、合併結果のカラム名になります。
- ORDER BYは、最後のSELECTで指定します。順序付けるカラムは、SELECTカラムリストのカラム番号で指定します。

➲ **使用例1　UNION**オペレータの使用例：

```
SELECT C1, C2 FROM T1
       UNION
SELECT C3, C4 FROM T2
           ORDER BY 2;
```

UNIONオペレータで合併した結果セットの重複行は取り除かれ、異なる値の行だけになります。重複行を残すときは、UNION ALLキーワードを使用します。UNION ALLは、個々の結果セットの行をそのまま残し、UNIONオペレータよりも高速になります。

➲ **使用例2　UNION ALL**の使用例：

```
SELECT 'MOVIE', Event FROM Entertainment WHERE Type = 'MOVIE'
        UNION ALL
SELECT 'BOOK', Name FROM MyBook;
```

副問合せ

副問合せは、WHEREやHAVINGの中に現れる問合せです。副問合せは必ず括弧で括ります。その他はSELECT文と同じ形式です。

➲ **使用例**　従業員給与の平均値を求める副問合せを使用して、平均以上の給与をもつ従業員名を全てリストします：

```
SELECT Name FROM Employee
       WHERE Salary > (SELECT AVG(Salary) FROM Employee);
```

副問合せの結果セットは1カラムでなければなりません。更に、問合せの結果を単純な比較演算に使用するときは、1行だけの値が生成されなければなりません。

IN副問合せ

IN副問合せは、メンバーかどうかをテストします。式の値が副問合せで検索した結果値のどれかと一致すると、真になります。IN副問合せは、複数行の1カラムデータを返すことができます。

➲ **使用例**　部門がNY市の何処かにある従業員名を全て検索します：

```
SELECT Name FROM Employee
       WHERE Dept_Id
          IN (SELECT Dept_Id FROM Department WHERE City = 'NY'');
```

EXISTS 副問合せ

EXIST副問合せは、結果セットが空かどうかをテストします。結果セットに行があれば、真になります。

➲ **使用例**　少なくとも一人は**Salary**が**$500,000**を超える従業員がいる部門名を全てリストします：

```
SELECT Dept_Name FROM Department d
       WHERE EXISTS
             (SELECT Dept_Id FROM EMPLOYEE e
                    WHERE e.Salary > 500000 AND d.Dept_Id = e.Dept_Id);
```

ときには、副問合せの中で、主問合せの現在行の値を使用しなければならないことがあります。これを外参照といいます。上の使用例の**d**.**Dept_id**カラムは外参照です。副問合せが入れ子になり、外側の副問合せの表カラムを外参照することもあります。

ANY／ALL／SOME副問合せ

ALL副問合せは、全ての結果値に対して比較演算が真のときに検索条件を真にします。副問合せの結果セットが空のときは、検索条件は真になります。結果セットの中にNULLがあると、検索条件は偽になります。

ANY副問合せは、少なくとも一つの結果値に対して比較演算が真のときに検索条件を真にします。副問合せの結果セットが空のときは、検索条件は偽になります。SOME副問合せはANY副問合せと同じです。

➲ **使用例**　マネージャでない従業員で**Salary**がマネージャの誰かよりも高い従業員名を全てリストします：

```
SELECT Emp_Name FROM Employee
       WHERE Manager = 'N' AND Salary > ANY
            (SELECT Salary FROM EMPLOYEE WHERE Manager = 'Y');
```

FOR BROWSE句

FOR BROWSEキーワードは、何もロックしないブラウズモードの検索を指示します。ブラウズモードでは、他のユーザーの検索をブロックすることはありません。何もロックしないので、読み込みの「繰り返し可能性」は保証されません。ブラウズモードは、データを表示したり報告書を作成するときに役に立ちます。

LIMIT句

オフセット　　結果セットの最初の戻り行からのオフセット

行　　戻り行の数

LIMITキーワードは、結果セット全体のオフセット値から戻すレコードの数を指示します。

➲ **使用例**

```
SELECT * FROM tb_test ORDER BY c1 LIMIT 10;
```

集合関数

集合関数は一セットの入力値からシングル結果を計算します。DBMaster は以下の集合関数をサポートします：

- MIN
- MAX
- AVG
- COUNT
- SUM
- XMLAGG

MIN関数は全ての入力値の最小値を返します。

MAX関数は全ての入力値の最大値を返します。

AVG関数は表現式の平均値(算数型)を返します。

COUNT 関数は設定標準に適応するレコードの数を返します。

SUM 関数は全ての入力値の総合を返します。

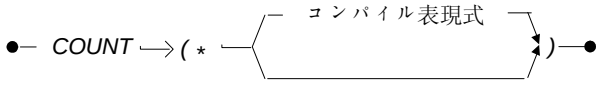
XMLAGG 関数は連鎖のXML値を返します。

構文は以下のようです：

```
{AVG|MAX|MIN|SUM|XMLAGG} ([ALL|DISTINCT] expression
[,comparsion-predicate])
|COUNT (* [,comparsion-predicate])
```

```
|COUNT ([ALL|DISTINCT] expression [,comparsion-predicate])
```

- Comparison-predicate……. コンパイル表現式.



➲ **使用例**

```
select count(*) FROM tb_test;
```

集合関数はSELECTコマンドのHAVING句、GROUP BY句、ORDER BY句に表現式として使用されますが、WHEREようなほかの句に使用できません。

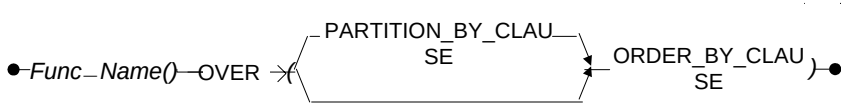
ウィンドウ関数

ウィンドウ関数は表の中に今の行に関する行セットの計算を実行します。これは集合関数の計算タイプとほぼ同じです。区別はウィンドウ関数の戻り値は行がシングルの行として出力できないことです。

```
func_name() OVER ([PARTITION_BY_CLAUSE] ORDER_BY_CLAUSE)
```

PARTITION_BY_CLAUSE　　結果セットにパーティションを分割するカラムです。ウィンドウ関数は各パーティションに適用され、再計算します。

ORDER_BY_CLAUSE　　ウィンドウ関数オーダーに適用するカラムを指定します。



DBMaster 以下のウィンドウ関数をサポートします：

- ROW_NUMBER
- RANK
- DENSE_RANK

ROW_NUMBER関数は結果セットのパーティションに一連の行数を返します。パーティションの第一行に1からはじめます。返すタイプはBIGINTです。

RANK関数はDENSE_RANK関数と似ています、一グループの値に各値のランクを返すため使用されます。区別は、テスト値が同じになると、RANK関数は連続できないランクを返しますが、DENSE_RANK関数は連続ランクを返すことです。RANK関数とDENSE_RANK関数の返り値のタイプはBIGINTです。

➲ 使用例

ブックカテゴリに販売量のNUMBER、RANKとDENSE_RANKは以下の例に示しますrank：

```
      Select TITLE, BOOK_CATEGORY, SALE_QTY,
      ROW_NUMBER() OVER (PARTITION BY BOOK_CATEGORY ORDER BY SALE_QTY) AS
ROW_NUMBER,
      RANK() OVER (PARTITION BY BOOK_CATEGORY ORDER BY SALE_QTY) AS RANK,
      DENSE_RANK() OVER (PARTITION BY BOOK_CATEGORY ORDER BY SALE_QTY) AS
DENSE_RANK
      FROM BOOK_STORE;
```

以下は結果です：

```
TITLE    CATEGORY SALE_QTY     ROW_NUMBER          RANK          DENSE_RANK
======== ======== ======== ================ ================ ================

book3    business     20          1               1               1
```

```
book2    business    30          2               2               2
book1    business    40          3               3               3
book1    computer    10          1               1               1
book2    computer    20          2               2               2
book3    computer    20          3               2               2
book4    computer    30          4               4               3
```

三つのウィンドウ 関数を使用する場合の制限があります：

- ORDER BY CLAUSEは order byを使用することができません

➲ **使用例**

```
select row_number() over (order by 1) from t1;
```

- OVER 句はクエリに使用するとすべてのウィンドウ 関数と同じに無ければなりません。

➲ **使用例**

```
Select row_number() over (order by c1), row_number() over (order by c2) from
t1;
```

- GROUP BY 或いは集合関数でのウィンドウ 関数はサポートできません。

➲ **使用例**

```
Select row_number() over (order by c1), row_number() over (order by c2) from
t1;
```

- INSERT, DELETEまたはUPDATEはウィンドウ関数をサポートできません。
- WHERE 句或いはサブクエリでのウィンドウ関数はサポートできません。

XML関数

XML関数はSQLデータからxml内容を生成する一セットの関数です。

DBMasterはXML関数をサポートします、これはSQL声明の一部分です。ユーザーはdmsql、odbc或いはjdbcインターフェースを通ってこの関数を使用することができます。

DBMaster以下のXML関数がサポートできます。

- xmlelement
- xmlforest
- xmlagg(xml)
- xmlcomment(text)

xmlelement構文は以下です：

```
xmlelement(name name [, xmlattributes(value AS attname [, ... ])] [,
content, ...])
```

xmlelement　　表現式は指定名、属性、内容を含むxml要素を生成します。

name　xml 要素のタグ名。名は無効な名のキャラクタを含むと、hexフォーマットを使用して代わります。例えば、名は 'phone number'(ここにはphoneとnumberの間にスペースがあります)、タグ名はにphone_x20_number代わります。

attname 属性の名

content 純テキスト、サブxml要素、或いはxml コメント

➲ **使用例**

```
dmSQL> SELECT xmlelement(name foo, xmlattributes(current_date as bar),
'cont', 'ent');


XMLELEMENT(NAME FOO, XMLATTRIBUTES(CURRENT_DATE AS BAR), 'CONT', 'ENT')
================================================================================

<foo bar="2011-08-18">content</foo>


1 rows selected
```

以下のは xmlforest構文です：

```
xmlforest(content [AS name] [, ...])
```

xmlforest表現式は指定した名と内容を使用する要素のXML フォレスト（シーケンス）を生成します。名が指定しなくて、コンテント値はカラム参照にすると、ディフォルトはカラム名になります。

➲ 使用例

```
dmSQL> select xmlforest(empname, phone) from employee;


XMLFOREST(EMPNAME, PHONE)

================================================================================

<empname>Abby</empname><phone>123-1234</phone>

<empname>Alice</empname><phone>234-1234</phone>
<empname>Amber</empname><phone>567-1234</phone>

3 rows selected
```

以下のはxmlagg(xml)の構文です：

```
xmlagg(xml)
```

xmlaggは集合関数です。これは行の入力値を連結します。Xmlaggの入力はxmlフラグメントにする必要があります。出力はCLOBタイプです。コンテントがないと、xml 要素は<ABC/>のような空の要素として表示されます。タグを起動/終了した後、余分な新規ラインを追加することができません。

➲ 使用例

```
dmSQL> select xmlagg(xmlelement(name person, xmlelement(name name, empname)))
from employee;


XMLAGG(XMLELEMENT(NAME PERSON, XMLELEMENT(NAME NAME, EMPNAME)))

================================================================================
<person><name>Abby</name></person><person><name>Alice</name></person><person>
<name>Amber</name></person>


1 rows selected
```

xmlcomment(text)の構文は以下です。

xmlcomment(text)

XMLCOMMENT はudfです。入力はnchar 或いは char データを生成するsql表現式です。出力は<!—から始めて-->まで終了するXMLコメント型にストリングです。

text　　textはエスケープキャラクタ(eg, < > &)を含むと、このキャラクタは実体で表示します。

➲ 使用例

```
dmSQL> select xmlcomment(empname) from employee;


XMLCOMMENT(EMPNAME)
================================================================================
<!--Abby-->
<!--Alice-->
<!--Amber-->


3 rows selected
```

XML 関数を使用する場合、四つの制限があります：

- 自動に入力をcharとして転換します。UDFの入力タイプが予め定義されていますからです。
- XMLAGGを除いて、ほかの関数の出力値はcharタイプです。サイズは制限になります（ページのサイズと関係がある）。例えば、DB_PGSIZ = 8になると、出力ストリングサイズは8056を超えません、巨大なデータは警告情報が無くて切り詰められます。
- 入力値はncharタイプになると、キャラクタはLCODEタイプに転換できない場合、無効に出力になるかもしれません。例えば、原始データはncharタイプのカラムにストアドすると、伝統中国語キャラクタと日本語キャラクタを含みます、LCODEの値を2に設定して伝統中国語キャラクタは適当に転換できません。

- タグが終了する時、新規なラインが追加しません。特別なxml フォーマットをサポートしません。

関連SQL

CREATE TABLE

CREATE VIEW

DELETE

INSERT

LOCK TABLE

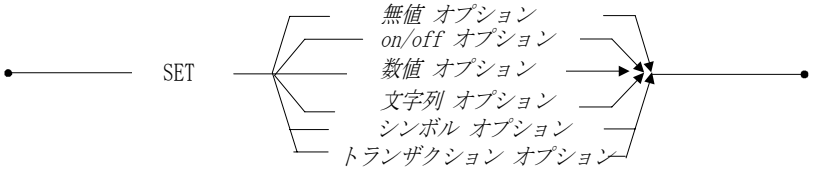
UPDATE

3.84 SET CONNECTION OPTIONS

目的

ODBCの各種接続オプションを設定します。

構文



無値オプション	値を指定しないオプション
on/off オプション	*on／off* を指定するオプション
文字列オプション	*'FOB'* のように引用符で囲われた文字列を指定するオプション

数値オプション	整数値を指定するオプション
シンボルオプション	*{delete | close | preserve}*のようなシンボル値を指定するオプション
トランザクションオプション	接続オプションを指定するオプション

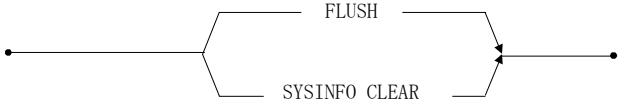
説明

SET文は、各種の接続オプションを設定するためのSQL構文です。Delphiのように、ODBC接続ハンドルをもたずにデータベースに接続するフロントエンドツールを使用するユーザーに役に立ちます。SQL文で必要な接続オプションを直接設定することができます。

以下にSET文で設定することができる全てのオプションを詳細に説明します。オプションは、パラメータのタイプによって*無値オプション*、*ON/OFFオプション*、*数値オプション*、*文字列オプション*、*シンボルオプション*の5つのグループに分けられます。

無値オプション

このグループのオプションは、パラメータのない単純なコマンドです。



SET FLUSH

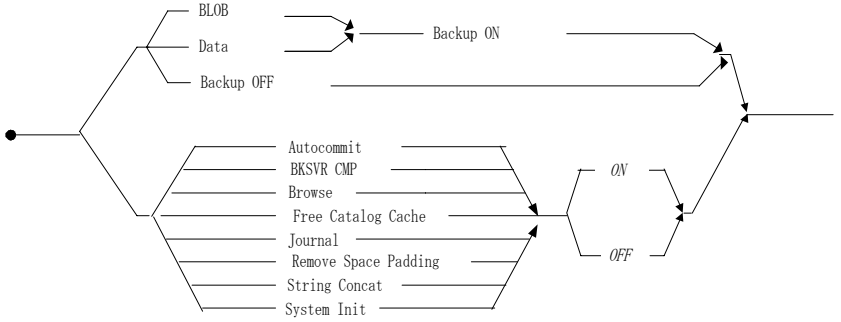
レプリケーションサーバーのオプションです。ターゲット側(複数)へのレプリケーションをフラッシュします。

SET SYSINFO CLEAR

SYSINFOシステム表をリセットしシステム情報をクリアします。

ON／OFFオプション

このグループでは、全てのオプションの有効値は*ON*または*OFF*です。*ON*／*OFF*の一方しかないオプションもあります。



SET AUTOCOMMIT on／off

自動コミットを*ON*／*OFF*にします。

SET BACKUP off

バックアップ・モードを*OFF*にします。DB_BMODEを0に設定するのと同じになります。

SET BKSVR CMP on／off

バックアップサーバーのコンパクトバックアップを*ON*／*OFF*にします。

SET BLOB BACKUP on

バックアップ・モードを BACKUP-DATA-AND-BLOB にします。DB_BMODEを2に設定するのと同じになります。

SET BROWSE on／off

SQL_TXN_READ_UNCOMMITTED (*ON*)またはSQL_TXN_SERIALIZABLE (*OFF*)をSQL_ATTR_TXN_ISOLATION接続オプションに設定します。詳細は、ODBCプログラミングガイドにある関数「SQLGetInfo」のオプション「SQL_DEFAULT_TXN_ISOLATION」を参照してください。

SET DATA BACKUP on

バックアップ・モードをBACKUP-DATAにします。DB_BMODEを1に設定するのと同じになります。

SET FREE CATALOG CACHE on／off

システムカタログキャッシュをフリー(*ON*)／保存 (*OFF*)します。

SET JOURNAL on／off

DBAだけがジャーナル書き出しを*ON*／*OFF*にすることができます。

SET REMOVE SPACE PADDING on／off

文字列データの後にある空白を自動的に取り除く機能を*ON*／*OFF*にします。

SET STRING CONCAT on／off

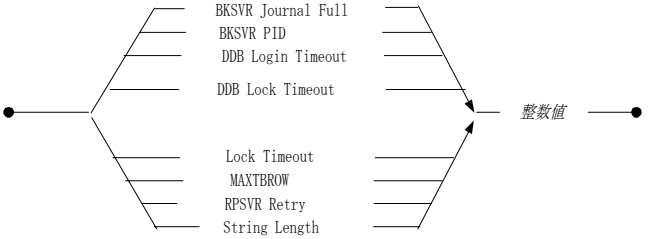
文字列連結の演算子(||)のオプションです。*ON*にすると、連結する前にCHARデータに埋め込まれた空白を削除して連結します。*OFF*にすると、埋め込まれた空白をそのまま残します。

SET SYSTEM INIT on／off

DBAだけがシステムモードを*ON*／*OFF*することができます。システムモードでは、システム表を作成します。

数値オプション

整数値を指定するオプションのグループです。各オプションには有効値の範囲があります。



SET BKSVR JOURNAL FULL 整数値

バックアップサーバーのジャーナルフルのパーセントを0～100に設定します。

SET BKSVR PID整数値

バックアップサーバーのプロセスIDを設定します。現状では0にします。

SET DDB LOGIN TIMEOUT整数値

DDB接続のログインタイムアウト時間を設定します。

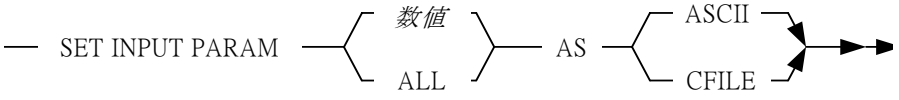
SET DDB LOCK TIMEOUT整数値

DDB接続のロック・タイムアウト時間を設定します。

SET INPUT PARAM n AS CFILE | ASCII

このセット・オプションは、パラメーターを使用するINSERTまたはUPDATEのステートメントの前に使用されます。ユーザがクライアント・ファイルにステートメントでのパラメーターの1つ以上をバインドしたければ、それが使用されます。次のステートメントでの対応するパラメーターかパラメーター用の入力データは、クライアント・ファイルに結び付けられるでしょう。挿入するべきデータは文字タイプ・データでなければなりません。また、パラメーターはLONG VARCHARあるいはLONG VARBINARYタイプ・カラムのいずれかでなければなりません。

ALLオプションを使用してクライアント・ファイルにパラメーターをすべてバインドしてください。CFILEオプションをクライアント・ファイルにパラメーターをバインドするために使用しなければなりません。クライアント・ファイルにパラメーターを結び付けないようDBMasterをリセットするためには、ASCIIオプションを備えたSET INPUT PARAMステートメントを使用してください。



数値　　クライアント・ファイルに順にどのパラメーターをバインドしなけなければならないか明示します。

➲ **使用例**

この例において、ファイル「dmconfig.ini」は、ホスト変数を使用して**c3**カラムに挿入することができます。

```
CREATE TABLE tb_attri (c1 INT, c2 INT, c3 LONG VARBINARY);
SET INPUT PARAM 3 AS CFILE;
INSERT INTO f1 VALUES (?,?,?);
2,2,'dmconfig.ini';
end;
```

SET LOCK TIMEOUT整数値

ロック・タイムアウトの秒数を設定します。0にすると待たずにアプリケーションに戻り、負の整数にするとロック解除を待ち続けます。

SET MAXTBROW整数値

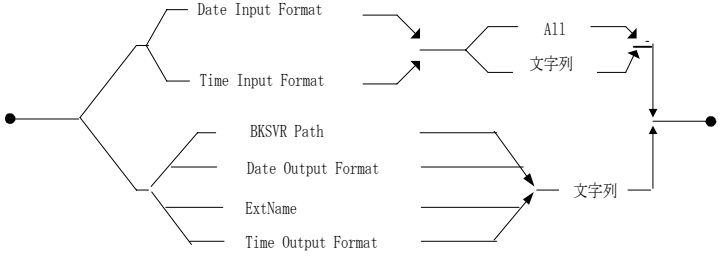
表検索で返す最大データ行数を設定します。0または負数を設定すると、全ての検索行が返されます。

SET RPSVR RETRY整数値

レプリケーション中にネットワーク障害が発生したときの再試行回数を設定します。

文字列オプション

このグループのオプションは引用符で囲われた文字列を指定します。文字列のフォーマットが決められているオプションもあります。



SET BKSVR PATH文字列

バックアップ・ジャーナル・ファイルのパスを設定します。

SET DATE INPUT FORMAT {ALL | 文字列 }

DATEカラムの入力フォーマットを設定します。ALLは全ての入力フォーマットの日付を入力可能にします。

日付の入出力フォーマット：

フォーマット	例
‘mm/dd/yy’	‘02/18/99’
‘mm-dd-yy’	‘02-18-99’
‘dd-mon-yy’	‘18-Feb-99’
‘mm/dd/yyyy’	‘02/18/1999’
‘mm-dd-yyyy’	‘02-18-1999’
‘yyyy/mm/dd’	‘1999/02/18’

‘yyyy-mm-dd’	‘1999-02-18’
‘dd/mon/yyyy’	‘18/Feb/1999’
‘dd-mon-yyyy’	‘18-Feb-1999’
‘dd.mm.yyyy’	‘18.2.1999’

(*yy/yyyy：年*、*mm：月*、*dd：日*)

SET DATE OUTPUT FORMAT文字列

DATEカラムの日付出力フォーマットを設定します。日付の入出力フォーマット表を参照してください。

SET EXTNAME TO文字列

サーバーファイルオブジェクトの拡張子の文字列を設定します。

SET TIME INPUT FORMAT { ALL | 文字列 }

TIMEカラムの時刻入力フォーマットを設定します。ALLは全ての入力フォーマットの時刻を入力可能にします。

時刻の入出力フォーマット：

フォーマット	例
‘hh:mm:ss.fff’	22:10:20.30
‘hh:mm:ss’	22:10:20
‘hh:mm	22:10
‘hh’	22
‘hh:mm:ss.fff tt’	10:10:20.30 PM
‘hh:mm:ss tt’	10:10:20 PM
‘hh:mm tt’	10:10 PM
‘hh tt’	10 PM
‘tt hh:mm:ss.fff’	PM 10:10:20.30
‘tt hh:mm:ss’	PM 10:10:20

‘tt hh:mm’	PM 10:10
‘tt hh’	PM 10

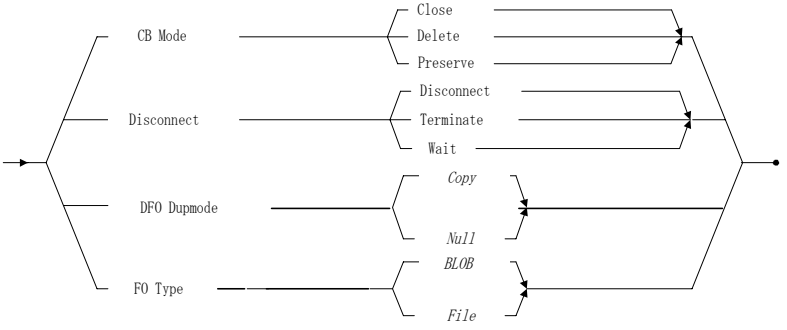
(*hh：時*、*mm：分*、*ss：秒*、*fff：小数部*、*tt：* AM／PM)

SET TIME OUTPUT FORMAT文字列

TIMEカラムの時刻出力フォーマットを設定します。時刻の入出力フォーマット表を参照してください。

シンボルオプション

このグループのオプションは、主にODBCシンボルにマッチするシンボルを設定します。詳細については、対応するODBC接続オプションを参照してください。



SET CB MODE { close | delete | preserve }

トランザクションがコミットされたときのカーソルの振る舞いを設定します。三つのモードの詳細については、ODBCプログラマガイドにあるSQLGetInfo関数のSQL_CURSOR_COMMIT_BEHAVIORオプションを参照してください。

SET CONCAT NULL RETURN { NULL | STRING }

このオプションは内蔵されたCONCAT関数あるいは連結オペレーター(||)の空ストリング連結時のために使用されます。このオプションのデフォルトはNULLです。このオプションがNULLにセットされる場合、NULLの値と連結されたどんなストリングもNULLで返るでしょう。オプションがSTRINGにセットされる場合、NULLの値が空ストリングとして扱われるので、NULLの値と連結されたどんなストリングもストリングを返すでしょう。

SET DISCONNECT { disconnect | terminat | wait }

SQLDisconnect()関数のアクションを設定します。DISCONNECTはサーバーから切断します。TERMOINATEはデータベースをシャットダウンします。WAITはサーバーが完全にシャットダウンされるのを待ちます。WAITオプションはDBMasterの内部オプションです。SQLDisconnect()を呼び出してデータベースをシャットダウンするツールの開発に使用します。

SET DFO DUPMODE { copy | null }

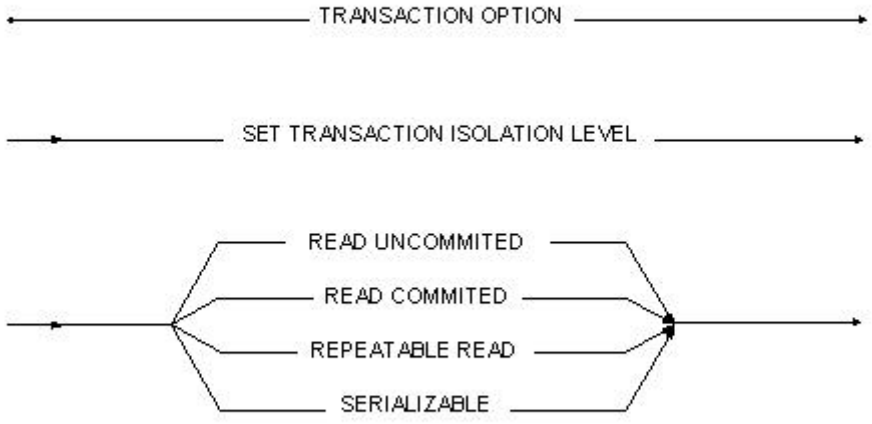
「SELECT INFO」を実行したときに、リモート表のFILEカラムにファイルオブジェクトをコピーするかどうかを指定するオプションです。NULLにするとFILEカラムがNULLになります。その他はリモートファイルオブジェクトをローカル表にコピーします。

SET FO TYPE { blob | file }

FILEカラムにマップするデータ型を選択します。FILEはFILEタイプ、その他はLONGVARBINARYタイプにします。

トランザクション・オプション

このグループのオプションは、接続オプションSQL_ATTR_TXN_ISOLATIONを、SQL_TXN_READ_UNCOMMITTED (*ON*)、或いはSQL_TXN_SERIALIZABLE (*OFF*)にセットします。詳細については、「ODBCプログラマーガイド」にあるSQLGetInfo関数のSQL_DEFAULT_TXN_ISOLATIONオプションを参照して下さい。



使用例

SET文の使用例を以下に示します。

➲ **使用例1　SET BKSVR PID：**

```
SET BKSVR PID 0;
```

➲ **使用例2　SET BKSVR PATH：**

```
SET BKSVR PATH ‘d:\data\backup’;
```

➲ **使用例3　SET DATE INPUT FORMAT：**

```
SET DATE INPUT FORMAT ALL;
SET DATE INPUT FORMAT ‘yyyy/mm/dd’;
```

➲ **使用例4　SET DATE OUTPUT FORMAT：**

```
SET DATE OUTPUT FORMAT ‘mm-dd-yy’;// DATE カラムは 12-31-99 のように表示されます
```

➲ **使用例5　SET DDB LOCK TIMEOUT：**

```
SET DDB LOCK TIMEOUT 20;          // ロック・タイムアウトは 20 秒です
```

➲ **使用例6　SET DDB LOGIN TIMEOUT：**

```
SET DDB LOGIN TIMEOUT 15;
```

➲ **使用例7**　使用例**7～10**は**SET DFO DUPMOD**Eの例です。データベース**db1**と**db2**にある表t1を使用します。**t1**は次のように定義されているとします：

```
CREATE TABLE t1 (c1 INT, c2 FILE);
```

➲ **使用例8**　**db1**のリモートデータベースとして**db2**を使用し**SET DFO DUPMODE**を**NULL**に設定します：

```
SET DFO DUPMODE null;
```

➲ **使用例9**　表**db2:t1**から検索したデータ行をt1に挿入します。表**t1**のカラム**c2**は**NULL**です：

```
SELECT c1, c2 from DB2:SYSADM.t1 INTO t1;
```

➲ **使用例10　SET DFO DUPMODE**を**COPY**に設定します。表**db2:t1**から検索したデータ行を**t1**に挿入すると、カラム**c2**に**db2:t1**のカラムc2がコピーされます：

```
SET DFO DUPMODE copy;
```

➲ **使用例11　SET EXTNAME TO**でサーバーファイルオブジェクトの拡張子を‘**.FOB**’にします：

```
SET EXTNAME TO ‘FOB’;
```

➲ **使用例12　SET LOCK TIMEOUT：**

```
SET LOCK TIMEOUT 30;              // タイムアウトは 30 秒です
SET LOCK TIMEOUT 0;               // 待たずに直ちに戻ります
SET LOCK TIMEOUT –5;              // 待ち続けます
```

➲ **使用例13　SET MAXTBROW：**

```
SET MAXTBROW 10;                  // 最初の 10 行だけ検索データを返します
```

```
SET MAXTBROW -3;                    // 全ての検索データを返します
```

➲ **使用例14　SET SYSTEM INIT：**

```
SET SYSTEM INIT ON;
CREATE TABLE SYSTEM.t1 (c1 int);
```

➲ **使用例15　SET TIME INPUT FORMAT：**

```
SET TIME INPUT FORMAT ALL;        // 全てのフォーマットを使用可能にします
SET TIME INPUT FORMAT 'hh:mm';    // 10:20 のように時刻を入力します
```

➲ **使用例16　SET TIME OUTPUT FORMAT：**

```
SET TIME OUTPUT FORMAT 'hh:mm:ss';        // 10:20:55 のように時刻を出力します
```

関連SQL

BEGIN BACKUP

END BACKUP (SET BKSVR …)

COMMIT WORK (SET CB MODE)

CREATE TABLESPACE (SET BACKUP OFF、 SET BLOB BACKUP ON、SET DATA BACKUP ON)

DISCONNECT (SET DISCONNECT)

INSERT (SET EXTNAME TO、SET DATE/TIME INPUT FORMAT)

SELECT (SET BROWSE、SET DFO DUPMODE、SET MAXTBROW、SET DATE/TIME OUTPUT FORMAT)

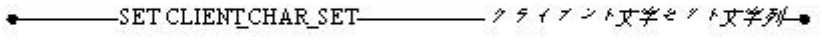
TERMINATE DATAGBASE (SET DISCONNECT)

3.85　SET CLIENT_CHAR_SET

目的

データベースのクライアント側に文字セットを定義します。

構文



クライアント文字セット文字列	クライアント側の文字セットを設定することができます

ASCII(英語)
BIG5 (繁体中国語)
Shift-JIS (日本語 Shift-JIS + Half Corner)
GBK (簡体中国語)
ISO-8859-1 (ラテン1 コード)
ISO-8859-2 (ラテン2 コード)
ISO-8859-5 (キリル コード)
ISO-8859-7 (ギリシアコード)
EUC-JP (日本語コード)
GB18030 (簡体中国語)
UTF-8 (UTF-8)

ISO-8859-{3,4,9,10,13,14,15,16}、KOI8-R、KOI8-U、KOI8-RU、CP{1250,1251,1252,1253,1254,1257}、CP{850,866}、Mac{ローマ語、中央ヨーロッパ語、アイスランド語、クロアチア語、ルーマニア語}、Mac{キリル語、ウクライナ語、ギリシア語、トルコ語 }、Macintosh(ヨーロッパ言語)

ISO-8859-{6,8}、CP{1255,1256}、CP862 、Mac{ヘブライ語、アラビア語}(セム言語)

CP932、ISO-2022-JP、ISO-2022-JP-2、ISO-2022-JP-1(日本語)

EUC-CN、CP936 、EUC-TW、CP950(中国語)

EUC-KR、CP949、JOHAB(韓国語)

Georgian-Academy、Georgian-PS(グルジア語)

KOI8-T(タジク語)

PT154(カザフ語)

TIS-620、CP874、MacThai(タイ語)

MuleLao-1、CP1133(ラオス語)
VISCII、TCVN、CP1258(ベトナム語)

説明

多言語データベースではクライアント側に複数のローカルコードはUTF-8データベースに接続します。クライアント側ではサーバ側で識別されるそれぞれの文字セットを定義します。クライアント側のdmconfig.iniでDB_CLILCODEにて文字コードの設定があればサーバ側の言語コードはキーワードDB_LCODEによって定義されます。加えてdmSQL上でSET CLIENT_CHAR_SETコマンドを使用して同様にクライアント側の文字セットを定義することができます。

ユーザはデータベースサーバとクライアントに設定されている文字コードを把握していることが望ましく、 UDF GETSYSINFO() を使用することで設定状態を取得することができます。サーバ側の文字コード設定を返す場合はSELECT GETSYSINFO(‘LCODE’); クライアント側の文字セットを返すコマンドはSELECT GETSYSINFO(‘CLILCODE’);

使用例

➲ **使用例**　クライアントの文字セットを**BIG5**に設定**SET**

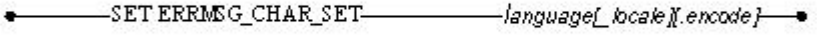
```
CLIENT_CHAR_SET ‘BIG5’;
```

3.86 SET ERRMSG_CHAR_SET

目的

SET ERRMSG_CHAR_SETコマンドはデータベースクライアント側の出力エラーメッセージの文字セットを定義します。

構文



有効値：

EN_XXX
EN_XXX.ASCII
EN_XXX.ISO-8859-1
EN_XXX.ISO-8859-2
EN_XXX.ISO-8859-5
EN_XXX.ISO-8859-7
EN_XXX.UTF-8
EN
EN.ASCII
EN.ISO-8859-1
EN.ISO-8859-2
EN.ISO-8859-5
EN.ISO-8859-7
EN.UTF-8
JA_XXX
JA_XXX.SHIFT-JIS
JA_XXX.SHIFT_JIS
JA_XXX.UTF-8
JA_XXX.EUCJP
JA_XXX.EUC-JP
JA
JA.SHIFT-JIS
JA.SHIFT_JIS
JA.UTF-8
JA.EUCJP
JA.EUC-JP

ZH_CN
ZH_CN.GBK
ZH_CN.UTF-8
ZH_CN.GB18030
ZH_TW
ZH_TW.BIG5
ZH_TW.UTF-8

説明

SET ERRMSG_CHAR_SETコマンドはデータベースクライアントのエラーメッセージ出力の文字セットを設定します。多言語データベースではクライアント側でそれぞれのエラーメッセージ用出力文字セットを設定できます。エラー表はディレクトリdbmaster/5.3/shared/locale/locale.lang/に置かれます。

コマンドは‘language*[_locale][.enlcode]*’このように指定されます。‘locale’ 文字列はISO-639に準拠、 ‘language’ はISO-3166標準です。一種類のエラー表のみ対応の場合、‘locale’または‘locale.utf8’を設定する必要があります。複数のロケール持つ言語では適切なロケールを設定する必要があります。zh_CNやzh_TWではzhとした場合無効となります。現状DBMasterのクライアントエラーメッセージに4言語で対応しています。クライアントエラーメッセージの文字セット設定取得はSELECT GETSYSINFO(‘ERRLCODE’);コマンドにて。

使用例

- **使用例1** クライアントのエラーメッセージ出力文字セットのロケールを'**ja**'に設定：

```
SET ERRMSG_CHAR_SET ‘ja’;
```

- **使用例2** クライアントのエラーメッセージ出力文字セットのロケールを'**ja**'に文字セットを‘**EUC_JP**’に設定：

```
SET ERRMSG_CHAR_SET ‘ja.EUC-JP’;
```

➲ **使用例3**　クライアントのエラーメッセージ出力文字セットのロケールを'**ja**'に文字セットを‘**UTF-8**'に設定：

```
SET ERRMSG_CHAR_SET ‘ja.UTF-8’;
```

3.87 SET TABLE STATISTICS

目的

このコマンドは各表の更新状態のモードとサンプルオプションを指定します。

構文

表名　　*表の名*

モード　　*表の更新状態のモード*

- -1: 設定がなく、dmconfig.iniを使用して設定(ディフォルト)
- 0: 閉じる
- 1: 開く
- 2: 開く、スマート

サンプル　　*表の更新状態のサンプル*

-1或いはNULL： 設定がなく、dmconfig.iniを使用して設定(ディフォルト)
- 1 ~ 100

説明

SET TABLE STATISTICS コマンドはdmconfig.iniより高い優先権を持ちます。そして各表のセット情報は表SYSTABLEにストアドされます。カラムUPD_STS_MODEは表の統計モードを保存して、カラムUPD_STS_SAMPLEは表の統計モードのサンプル率を保存します。

モードを-1を設定して、サンプルを任意の有効値に設定する場合、UPD_STS_MODE と UPD_STS_SAMPLEは-1になります。

モードを0.1或いは2を設定して、サンプルを-1或いはスペースに設定する場合、UPD_STS_MODEはモードの値になり、UPD_STS_SAMPLEはシステムオプションSTSSPの値になります。

使用例

➲ **使用例　dmconfig.ini** のキーワード **DB_STSSP = 80**、**jeff.tb_staff** は有効な表名を設定します：

```
dmSQL> SET TABLE STATISTICS jeff.tb_staff 1;
dmSQL> select TABLE_NAME, UPD_STS_MODE, UPD_STS_SAMPLE from SYSTABLE;
         TABLE_NAME      UPD_STS_MODE   UPD_STS_SAMPLE
====================== ============= ===============
JEFF.TB_STAFF                 1              80
1 rows selected;
```

3.88 SUSPEND SCHEDULE

目的

非同期表レプリケーションのスケジュールを一時停止します。

構文

ターゲット・データベース名　レプリケーション・スケジュールを一時停止するターゲット・データベースの名前

説明

SUSPEND SCHEDULE文は、非同期表レプリケーションのスケジュールを一時停止します。レプリケーション・スケジュールが再開されるまで、ソース・データベースはターゲット・データベースに接続しようとはしません。ソース表の所有者、DBA、SYSADMだけが実行することができます。

一時停止した非同期表レプリケーションのスケジュールを再開するにはRESUME SCHEDULE文を使用します。

使用例

➲ **使用例**　ターゲット・データベース**DivOneDb**へのレプリケーション・スケジュールを一時停止します：

```
SUSPEND SCHEDULE FOR REPLICATION TO DivOneDb;
```

関連SQL

ALTER SCHEDULE

CREATE REPLICATION

CREATE SCHEDULE

DROP REPLICATION

DROP SCHEDULE

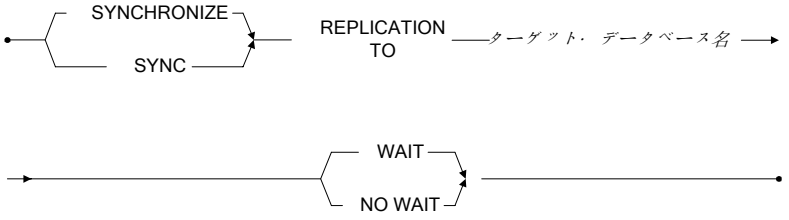
RESUME SCHEDULE

3.89 SYNCHRONIZE SCHEDULE

目的

次のレプリケーション・スケジュールの時点まで待たずに、ソース・データベースからターゲット・データベースへのデータを同期させます。

構文



ターゲット・データベース名　レプリケーションを同期させるターゲット・データベースの名前

説明

SYNCHRONIZE SCHEDULE文は、非同期レプリケーションで次のレプリケーション・スケジュールの時点まで待たずに、ターゲット・データベースの全てのデータをソース・データベースのデータに強制的に同期させます。ソース表の所有者、DBA、SYSADMだけが実行することができます。

WAITオプションは、ディストリビュータ・デーモンが全ての変更を終了するまで待機します。このコマンドは、レプリケーションの完了後にのみ戻ります。NO WAITオプションは、ディストリビュータ・デーモンにその作業を直ちに実行させ、SYNCコマンドが直ぐに戻されます。初期設定はWAITです。

使用例

➲ **使用例**　ターゲット・データベース**DivOneDb**へのレプリケーションを同期させます：

```
SYNCHRONIZE REPLICATION TO DivOneDb;
```

関連SQL

ALTER SCHEDULE

CREATE REPLICATION

CREATE SCHEDULE

DROP REPLICATION

DROP SCHEDULE

RESUME SCHEDULE

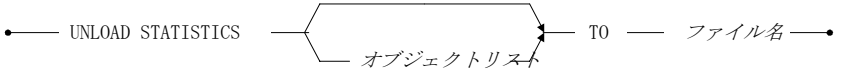
SUSPEND SCHEDULE

3.90 UNLOAD STATISTICS

目的

データベース統計をテキストファイルにアンロードします。

構文



オブジェクトリスト　統計データをアンロードするデータベースオブジェクトのリスト

ファイル名　　統計データを保存するテキストファイルの名前

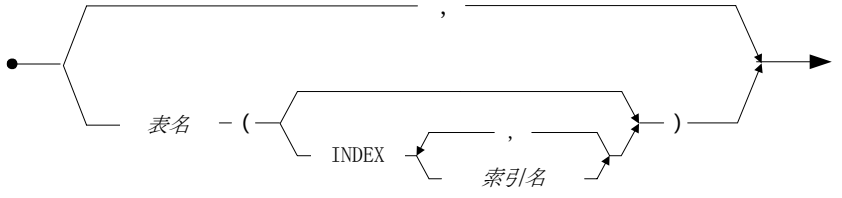
説明

UNLOAD STATISTICS文は、データベース統計をASCIIテキストファイルにアンロードします。ファイルを望みの統計データに編集しデータベースにロードします。DBAまたはSYSADMだけが実行することができます。

データベース全体または特定の表の統計情報をアンロードします。各表の表統計情報、カラム統計情報、索引統計情報、これらの組み合わせをアンロードするかどうかを指定します。

表統計は、表のページ数、行数、行の平均バイト数を記録します。カラム統計は、異なるカラム値の個数、カラムの平均・最小・最大バイト数を記録します。索引統計は、索引ページ数、索引ツリーのレベル数、リーフページ数、異なるキー値の個数、キー当たりのページ数、索引のクラスタカウントを記録します。

オブジェクトリスト



使用例

➲ **使用例**　データベース全体の統計をファイル**stat.dat**にアンロードします：

```
UNLOAD STATISTICS TO stat.dat;
```

関連SQL

LOAD STATISTICS

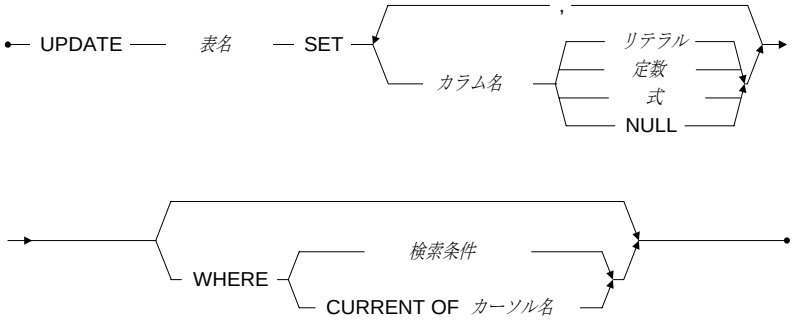
UPDATE STATISTICS

3.91 UPDATE

目的

表の指定カラムの値を更新します。

構文



表名	更新表の名前
カラム名	更新カラムの名前
リテラル	更新カラムのリテラル
式	更新カラムの値を返す式
定数	更新カラムの定数値
検索条件	更新する行の条件
カーソル名	位置付けUPDATEに使用するカーソルの名前 (ODBCプログラム内でのみ使用可能)

説明

UPDATE文は表行を更新します。システムカタログ表の行を更新することはできません。表所有者、DBA、SYSADM、表全体または特定のカラムのUPDATE権限をもつユーザーだけが実行することができます。

更新するカラムの新しい値は、カラム制約と参照整合性を満たさなければなりません。カラムに初期設定値を設定するときは、DEFAULTキーワードを使用します。

使用例

➲ **使用例1　Employeesinfo**表の従業員名**ChrisのSalary**を更新します：

```
UPDATE Employeesinfo SET Salary = 5000 WHERE FName = ‘Chris’;
```

➲ **使用例2　Employees**表の従業員名**ChrisのSalary**を**10%**上げて更新します：

```
UPDATE Employeesinfo SET Salary = Salary*1.10 WHERE FName = ‘Chris’;
```

関連SQL

CREATE TABLE

INSERT

LOCK TABLE

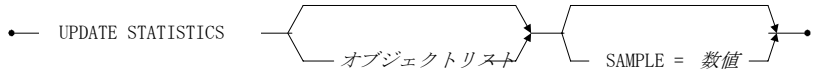
SELECT

3.92 UPDATE STATISTICS

目的

問合せを最適化するためにデータベース統計を更新します。

構文



オブジェクトリスト	統計データを更新するデータベースオブジェクトのリスト
数値	統計データの更新に使用するサンプルデータのパーセント

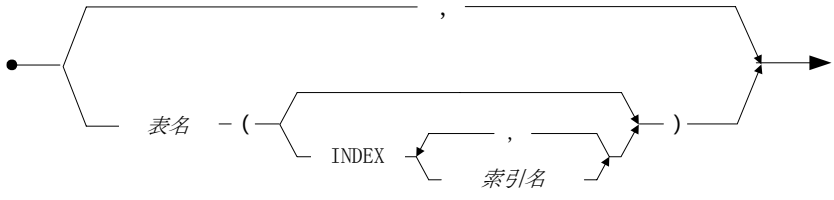
説明

UPDATE STATISTICS文は、データベース統計情報を更新します。統計情報を最新にしておくと、効率よく問合せを実行するのに役立ちます。オブジェクト所有者、DBA、SYSADMだけが実行することができます。

データベース全体あるいは特定の表の統計情報を更新します。各表の表統計情報、カラム統計情報、索引統計情報、これらの組み合わせを更新するかどうかを指定します。SAMPLEキーワードは、サンプルデータのパーセント(1～100)を指定します。

索引統計は、索引ページ数、索引ツリーのレベル数、リーフページ数、異なるキー値の個数、キー当たりのページ数、索引のクラスタカウントを記録します。

オブジェクトリスト



使用例

- **使用例1　30%**のサンプルを使用してデータベースの全ての統計情報を更新します：

```
UPDATE STATISTICS SAMPLE = 30;
```

- **使用例2　table1**の全ての統計情報を更新します：

```
UPDATE STATISTICS table1 SAMPLE = 50;
```

- **使用例3　table1**の索引**ix1**の統計情報を更新します：

```
UPDATE STATISTICS table1 (INDEX(ix1));
```

関連SQL

CREATE INDEX

CREATE TABLE

CREATE TABLESPACE

CREATE TEXT INDEX

LOAD STATISTICS

UNLOAD STATISTICS

UPDATE TABLESPACE STATISTICS

3.93 UPDATE TABLESPACE STATISTICS

目的

問合せを最適化するために表領域統計を更新します。

構文

オブジェクトリスト　統計データを更新する表領域オブジェクトのリスト

説明

UPDATE TABLESPACE STATISTICS文は、表領域統計情報を更新します。統計情報を最新にしておくと、効率よく問合せを実行するのに役立ちます。DBA、SYSADMだけが実行することができます。

表領域統計とファイル統計が更新されます。

表領域統計には、ページ数、フリーページ数、フレーム数、フリーフレーム数が記録されます。

ファイル統計には、ページ数／フレーム数、フリーページ数／フリーフレーム数が記録されます。

使用例

➲ **使用例**　表領域 **DEFTABLESPACE** の統計情報を更新します：

```
UPDATE TABLESPACE STATISTICS DEFTABLESPACE;
```

関連SQL

CREATE TABLESPACE

LOAD STATISTICS

UPDATE STATISTICS

UNLOAD STATISTICS

4 組み込み関数

DBMasterは多数の組み込み関数を提供しています。組み込み関数は、結果セットに入れる行のカラム値を指定したり、カラム値の条件を指定するのに使用されます。

組み込み関数は以下のカテゴリーに分けられます：

- 文字列関数
- 数値関数
- 日付関数と時刻関数
- システム関数

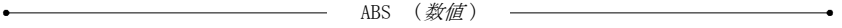
以下の各節では、各関数について説明します。各関数の引数と戻り値は、名前とデータ型と値からなる構文ダイアグラムの下で説明します。

4.1 ABS

目的

数値の絶対値を返します。

構文



数値	Double： 絶対値を求める数値
戻り値	Double： 数値の絶対値

説明

数値の絶対値を倍精度浮動小数点数で返します。

使用例

➲ **使用例**　戻り値は **3.14000000000000e+012** になります：

```
ABS(-3.14E12)
```

関連関数

SIGN

4.2 ACOS

目的

*アークコサイン*の値を返します。

構文

ACOS （ *数値* ）

数値　　Double： アークコサインを求める数値

戻り値　　Double：*数値*のアークコサイン

説明

数値のアークコサインを倍精度浮動小数点数で返します。数値の引数は0～πラジアンの範囲になります。

使用例

➲ **使用例**　戻り値は**1.04719755119660e+000**になります：

```
ACOS(0.5)
```

関連関数

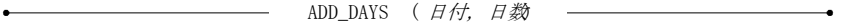
ASIN	COSH	SINH
ATAN	COT	TAN
ATAN2	DEGREES	TANH
COS	SIN	RADIANS

4.3 ADD_DAYS

目的

日付に日数を加えます。

構文



日付	Date： 日数を加える日付
日数	Integer： 日付に加える日数
戻り値	Date： *日付*に*日数*を加えた結果の日付

説明

日付に日数を加えた結果の日付を返します。負数の日数を与えることもできます。

使用例

➲ **使用例1**　日付**1999-03-01**を返します：

```
ADD_DAYS('1999-02-24', 5)
```

➲ **使用例2**　日付**2000-02-29**を返します：

```
ADD_DAYS('2000-02-24', 5)
```

関連関数

ADD_MONTHS

ADD_YEARS

ADD_HOURS

ADD_MINS

ADD_SECS

4.4 ADD_HOURS

目的

時刻に時間を加えます。

構文

ADD_HOURS （ 時刻,時間数 ）

時刻	Time： 時間を加える時刻
時間数	Integer： 時刻に加える時間数
戻り値	Time： *時刻* に *時間数* を加えた結果の時刻

説明

時刻に時間数を加えた結果の時刻を返します。負数の時間数を与えることもできます。

使用例

➲ **使用例1** 時刻**20：11：12**を返します：

```
ADD_HOURS('10:11:12', 10)
```

➲ **使用例2** 時刻**22：11：12**を返します：

```
ADD_HOURS('10:11:12', -12)
```

関連関数

ADD_DAYS

ADD_MONTHS

ADD_YEARS

ADD_MINS

ADD_SECS

4.5 ADD_MINS

目的

時刻に分を加えます。

構文

ADD_MINS （時刻, 分数）

時刻	Time： 分を加える時刻
分数	Integer： 時刻に加える分数
戻り値	Time：*時刻*に*分数*を加えた結果の時刻

説明

時刻に分数を加えた結果の時刻を返します。負数の分数を与えることもできます。

使用例

➲ **使用例1**　時刻**10：21：12**を返します：

```
ADD_MINS('10:11:12', 10)
```

➲ **使用例2**　時刻**09：59：12** を返します：

```
ADD_MINS('10:11:12', -12)
```

関連関数

ADD_DAYS

ADD_MONTHS

ADD_YEARS

ADD_HOURS

ADD_SECS

4.6 ADD_MONTHS

目的

日付に月数を加えます。

構文

ADD_MONTHS （ *日付, 月数*

日付	Date： 月数を加える日付
月数	Integer： 日付に加える月数
戻り値	Date：*日付*に*月数*を加えた結果の日付

説明

日付に月数を加えた結果の日付を返します。負数の月数を与えることもできます。

使用例

➲ **使用例1**　日付**1999-07-24**を返します：

```
ADD_MONTHS('1999-02-24',5)
```

➲ **使用例2**　日付**2001-01-01**を返します：

```
ADD_MONTHS('2000-01-01',12)
```

関連関数

ADD_DAYS

ADD_YEARS

ADD_HOURS

ADD_MINS

ADD_SECS

4.7 ADD_SECS

目的

時刻に秒数を加えます。

構文

ADD_SECS （ *時刻, 秒数*

時刻	Time： 秒を加える時刻
秒数	Integer： 時刻に加える秒数
戻り値	Time：*時刻* に *秒数* を加えた結果の時刻

説明

時刻に秒数を加えた結果の時刻を返します。負数の秒数を与えることもできます。

使用例

➲ **使用例1**　時刻**10：11：22**を返します：

```
ADD_SECS('10:11:12',10)
```

➲ **使用例2**　時刻**10：10：52**を返します

```
ADD_SECS('10:11:12', -20)
```

関連関数

ADD_DAYS

ADD_MONTHS

ADD_YEARS

ADD_HOURS

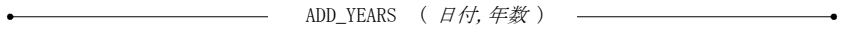
ADD_MINS

4.8 ADD_YEARS

目的

日付に年数を加えます。

構文



日付	Date： 年数を加える日付
年数	Integer： 日付に加える年数
戻り値	Date： *日付*に*年数*を加えた日付

説明

日付に年数を加えた結果の日付を返します。負数の年数を与えることもできます。

使用例

➲ **使用例1**　日付**2001-03-04**を返します：

```
ADD_YEARS('1999-03-04', 2)
```

➲ **使用例2**　日付**1995-02-28**を返します：

```
ADD_YEARS('2000-02-29', -5)
```

関連関数

ADD_DAYS

ADD_MONTHS

ADD_HOURS

ADD_MINS

ADD_SECS

4.9 ASCII

目的

先頭文字のASCIIコードを返します。

構文

文字列	String：先頭文字のASCIIコードを取得する文字列
戻り値	Integer：*文字列*の先頭文字のASCIIコード

説明

ASCII関数は、文字列の先頭文字のASCIIコード値を返します。文字列が空列のときは値0(NULL)が返されます。文字列を指定しないとエラーが返されます。

使用例

➲ **使用例1　"A"のASCIIコード65を返します：**

```
ASCII('A')
```

➲ **使用例2**　同様に**"A"のASCII**コード65を返します：

```
ASCII('ABC')
```

➲ **使用例3　"a"** の **ASCII** コード **97** を返します：

```
ASCII('a')
```

➲ **使用例4**　同様に **"a"** の **ASCII** コード **97** を返します：

```
ASCII('abc')
```

➲ **使用例5　"1"** の **ASCII** コード **49** を返します：

```
ASCII('1')
```

➲ **使用例6**　"**!**" の **ASCII** コード **33** を返します：

```
ASCII('!')
```

関連関数

CHAR

4.10 ASIN

目的

アークサインの値を返します。

構文

ASIN （*数値*）

数値	Double： アークサインを求める数値
戻り値	Double：*数値* のアークサイン

説明

数値のアークサインを倍精度浮動小数点数値で返します。戻り値は、**-π/2**～**π/2**の範囲になります。

使用例

➲ **使用例**　戻り値は **5.23598775598299e-001** になります：

```
ASIN(0.5)
```

関連関数

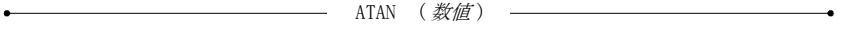
ACOS	COSH	SINH
ATAN	COT	TAN
ATAN2	DEGREES	TANH
COS	SIN	RADIANS

4.11 ATAN

目的

アークタンジェントの値を返します。

構文



数値	Double：アークタンジェントを求める数値
戻り値	Double：*数値* のアークタンジェント

説明

数値のアークタンジェントを倍精度浮動小数点数値で返します。戻り値は、**-π/2～π/2**の範囲になります。

使用例

➲ **使用例**　戻り値は **4.63647609000806e-001** になります：

```
ATAN(0.5)
```

関連関数

ACOS	COSH	SINH
ASIN	COT	TAN
ATAN2	DEGREES	TANH
COS	SIN	RADIANS

4.12 ATAN2

目的

アークタンジェント ***x/y*** の値を返します。

構文

ATAN2 （*x*, *y*）

x	Double： アークタンジェント ***x/y*** の分子
y	Double：アークタンジェント ***x/y*** の分母
戻り値	Double：アークタンジェント ***x/y*** の値

説明

アークタンジェント***x/y***の値を倍精度浮動小数点数値で返します。戻り値は-π～πの範囲になります。

使用例

➲ **使用例**　戻り値は **4.63647609000806e-001** になります：

```
ATAN2(0.1, 0.2)
```

関連関数

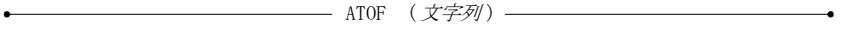
ACOS	COSH	SINH
ASIN	COT	TAN
ATAN	DEGREES	TANH
COS	SIN	RADIANS

4.13 ATOF

目的

ASCIIフォーマットの浮動小数点数を倍精度浮動小数点数に変換します。

構文



文字列	String ：倍精度浮動小数点数に変換する文字列
戻り値	Double：*文字列*の倍精度浮動小数点数値

説明

文字列の引数で表される数値の倍精度浮動小数点数値を返します。

使用例

➲ **使用例1　"-12.34"**の倍精度浮動小数点数値**-1.23400000000000e+001** を返します：

```
ATOF('-12.34')
```

➲ **使用例2　"-12.34 E34"**の倍精度浮動小数点数値**-1.23400000000000e+035** を返します：

```
ATOF('-12.34E34')
```

関連関数

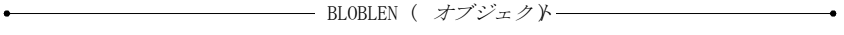
FTOA

4.14 BLOBLEN

目的

BLOBLEN関数は入力BLOBデータ長を返します。BLOBLENはサイズが2GBを超える場合でも、最大の(2GB-1)を返すので注意してください。BLOBLENはCLOB、BLOB 、NCLOB 、FILE タイプのデータ長を取得します。取られます。

構文



オブジェクト	Long varbinary： ソースBLOB
戻り値	Integer：ソースBLOBデータの長さ

注：　BLOBLENによって*long varchar*、*long varbinary*、*file*データ型のオブジェクトのデータ長を取得することもできます。

説明

BLOBデータの長さを返します。

使用例

➲ **使用例　BLOB「content」**のデータ長を返します：

```
BLOBLEN(content)
```

関連関数

HIGHLIGHT

HTMLHIGHLIGHT

HITCOUNT

HTIPOS

HTMLTITLE

4.15 BLOBLENEX

目的

BLOBデータの長さを十進数で返します。

構文

オブジェクト　BLOB: ソース BLOB

戻り値　Decimal : ソースBLOBデータの長さ

説明

BLOBLENEX 関数は入力 BLOB データ長を Decimal 値で返します。BLOBLENEXでは CLOB、BLOB、NCLOB 、FILEオブジェクト型のデータ長を取得できます。 BLOBLENEXはBLOBLENとは違い2GB以上でも正確なBLOBサイズを返します。

使用例

➲ **使用例**　BLOB “**content**”のデータ長を返します.

```
BLOBLENEX(content)
```

4.16 CEILING

目的

指定した数値より大きいか等しい直近の整数値を返します。

構文

数値	Double： 直近の天井整数を求める数値
戻り値	Double：*数値*以上の直近の整数値

説明

指定した数値より大きいか等しい直近の整数を倍精度浮動小数点数で返します。

使用例

➲ **使用例1　12.3**より大きい次の整数**1.30000000000000e+001**を返します：

```
CEILING(12.3)
```

➲ **使用例2　-12.3**より大きい次の整数**-1.20000000000000e+001**を返します：

```
CEILING(-12.3)
```

関連関数

FLOOR

FIX

RND

ROUND

MODFI

MODFM

4.17 CHAR

目的

ASCIIコードの文字を返します。

構文

CHAR （ *数値* ）

数値　　Integer：取得する文字のASCIIコード

戻り値　　String：*数値*のASCIIコードが表す文字

説明

CHAR関数は、数値で指定したASCIIコードの文字を返します。ASCIIコードは0～255の範囲になければなりません。他の値はASCIIコードではなく、不正または無効な値が返されます。*数値*の引数を指定しないとエラーになります。

使用例

➲ **使用例1　ASCII**コード**65**をもつ文字の文字列 “**A**” を返します：

```
CHAR(65)
```

➲ **使用例2　ASCII**コード**97**をもつ文字の文字列 “**a**” を返します：

```
CHAR(97)
```

➲ **使用例3　ASCII**コード**49**をもつ文字の文字列 "**1**" を返します：

```
CHAR(49)
```

➲ **使用例4　ASCII**コード**33**をもつ文字の文字列 "**!**" を返します：

```
CHAR(33)
```

関連関数

ASCII

4.18 CHAR_LENGTH

目的

文字列にある文字の数を返します。

構文

文字列	String: 長さを取得する文字列
戻り値	Integer: *文字列*にある文字の数

説明

文字列にある文字の数を返します。末尾の空白や文字列の端末文字は、除外されます。文字列の引数を与えないと、エラーになります。

使用例

➲ **使用例1**　数値「4」が返されます：

```
SELECT CHAR_LENGTH(' abc   ');
CHAR_LENGTH(' ABC   ')
```

```
=========================
          4
```

関連関数

CHARACTER_LENGTH

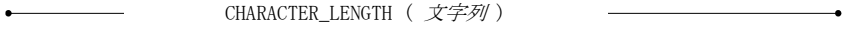
LENGTH

4.19 CHARACTER_LENGTH

目的

文字列にある文字の数を返します。

構文



文字列	String: 長さを取得する文字列
戻り値	Integer: *文字列*にある文字の数

説明

文字列にある文字の数を返します。末尾の空白や文字列の端末文字は、除外されます。文字列の引数を与えないと、エラーになります。

使用例

➲ **使用例**　数値「4」が返されます。

```
SELECT CHARACTER_LENGTH(' abc   ');
CHARACTER_LENGTH(' ABC   ')
=========================
```

4

関連関数

CHAR_LENGTH

LENGTH

4.20 CHECKMEDIAFORMAT

目的

BLOB が指定のメディア･フォーマットと一致したものかをチェックします

構文

CHECKMEDIATYPE (*blob*)

blob チェックを実行するカラム名

メディアフォーマット: 文字列: メディア･フォーマットを指定します。サポートフォーマット:DOC、 XLS、PPT 、HTM 、XML、PDF

戻り値: カラムにレコートがメディア･フォーマットとマッチすると正確です。

説明

CHECKMEDIAFORMAT関数は、blob が指定のメディアフォーマットとマッチするかをチェックします。

使用例

➲ **例:** blob カラムがDOC フォーマットとマッチするかをチェックします。

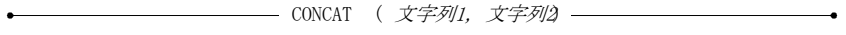
```
CHECKMEDIAFORMAT(wordcol, 'DOC')
```

4.21 CONCAT

目的

二つの文字列を連結します。

構文



文字列1	String：連結した文字列の先頭に置かれる文字列
文字列2	String：連結した文字列の末尾に置かれる文字列
戻り値	String：*文字列1* と *文字列2*の連結文字列

説明

CONCAT関数は、*文字列1*と *文字列2*を連結した文字列を返します。*文字列1*は連結した文字列の先頭に、*文字列2*は末尾に置かれます。両方の引数が与えられないとエラーになります。

引数がNULL値をもつときは、以下の文字列が返されます：

- 文字列1と文字列2が NULL のときは、NULL が返されます。
- 文字列1がNULLでなく文字列2がNULLのときは、文字列1が返されます。
- 文字列1がNULLで文字列2がNULLでないときは、文字列2が返されます。

使用例

➲ **使用例1**　文字列 “**master plan**” が返されます。最初の文字列の終わりにある空白に注意してください：

```
CONCAT('master ', 'plan')
```

➲ **使用例2**　文字列 “**mastermind**” が返されます：

```
CONCAT('master', 'mind')
```

関連関数

REPEAT

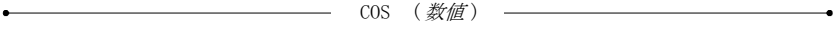
SPACE

4.22 COS

目的

コサインの値を返します。

構文



数値	Double： コサインを求める数値
戻り値	Double：*数値* のコサイン

説明

数値のコサインを返します。戻り値は、倍精度浮動小数点数のラジアンになります。

使用例

➲ **使用例**　戻り値は**8.77582561890373e-001** になります：

```
COS(0.5)
```

関連関数

ACOS	COSH	SINH
ASIN	COT	TAN
ATAN	DEGREES	TANH
ATAN2	SIN	RADIANS

4.23 COSH

目的

ハイパーボリック・コサインの値を返します。

構文

COSH （ *数値* ）

数値　　Double：ハイパーボリックコサインを求める数値

戻り値　　Double：*数値* のハイパーボリックコサイン

説明

数値のハイパーボリックコサインを返します。戻り値は、倍精度浮動小数点数のラジアンになります。

使用例

➲ **使用例**　戻り値は**1.12762596520638e+000** になります：

```
COSH(0.5)
```

関連関数

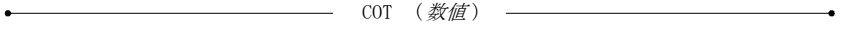
ACOS	COS	SINH
ASIN	COT	TAN
ATAN	DEGREES	TANH
ATAN2	SIN	RADIANS

4.24 COT

目的

コタンジェントの値を返します。

構文



数値	Double： コタンジェントを求める数値
戻り値	Double：*数値* のコタンジェント

説明

数値のコタンジェントを返します。戻り値は、倍精度浮動小数点数のラジアンになります。

使用例

➲ **使用例**　戻り値は**1.83048772171245e+000** になります：

```
COT(0.5)
```

関連関数

ACOS	COS	SINH
ASIN	COSH	TAN
ATAN	DEGREES	TANH
ATAN2	SIN	RADIANS

4.25 CURDATE

目的

現在の日付を返します。

構文

CURDATE ( )

戻り値　　Date：現在の日付

説明

現在の日付を返します。

使用例

➲ **使用例**　現在の日付を返します：

```
CURDATE()
```

関連関数

CURTIME

CURRENT_DATE

CURRENT_TIME

CURRENT_TIMESTAMP

NOW

4.26 CURRENT_DATE

目的

現在の日付を返します。

構文

戻り値　　Date：現在の日付

説明

DBMasterの初期設定日付出力フォーマットで、現在の日付を返します。

使用例

➲ **使用例1**　現在の日付を返します：

```
SELECT CURRENT_DATE;


CURRENT_DATE
=============
```

```
2001-09-26


1 rows selected
```

➲ **使用例2**　表**sql99t5**の**date**カラムに現在の日付を挿入し、そのデータを取り出します：

```
INSERT INTO sql99t5 VALUES (CURRENT_DATE);
1 row inserted


SELECT * FROM sql99t5;


   DATE
==========
2001-12-19
1 row selected
```

関連関数

CURRENT_TIME

CURRENT_TIMESTAMP

CURTIME

CURDATE

NOW

4.27 CURRENT_TIME

目的

現在の時刻を返します。

構文

戻り値　　TIME：現在の時刻

説明

DBMasterの初期設定時刻出力フォーマットで、現在の日付を返します。

使用例

➲ **使用例1**　現在の時刻を返します：

```
SELECT CURRENT_TIME;


CURRENT_TIME
==============
15:11:07


1 rows selected
```

➲ **使用例2**　表**sql99t4**の**time**カラムに現在の時刻を挿入し、そのデータを取り出します：

```
INSERT INTO sql99t4 VALUES (CURRENT_TIME);
1 row inserted


SELECT * FROM sql99t4;


   TIME
===========
15:36:41
```

```
1 row selected
```

関連関数

CURRENT_DATE

CURRENT_TIMESTAMP

CURDATE

CURTIME

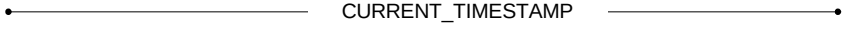
NOW

4.28 CURRENT_TIMESTAMP

目的

現在の日付と時刻を返します。

構文



戻り値　　TIMESTAMP：現在のタイムスタンプ

説明

DBMasterの初期設定タイムスタンプ出力フォーマットで、現在の日付と時刻を返します。

使用例

➲ **使用例1**　現在のタイムスタンプを返します：

```
SELECT CURRENT_TIMESTAMP;
```

```
     CURRENT_TIMESTAMP
==========================
2001-12-19 15:42:29


1 rows selected
```

➲ **使用例2**　表**sql99t3**の**date_time**カラムに現在の日付と時刻を挿入し、そのデータを取り出します：

```
INSERT INTO sql99t3 VALUES (CURRENT_TIMESTAMP);
1 row inserted


SELECT * FROM sql99t3;


   DATE_TIME
==========================
2001-12-19 15:42:29
1 row selected
```

関連関数

CURRENT_DATE

CURRENT_TIME

CURDATE

CURTIME

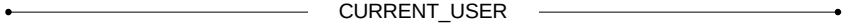
NOW

4.29 CURRENT_USER

目的

DBMasterに現在接続しているユーザーを返します。

構文



戻り値　　USER：現在のユーザー

説明

DBMasterに現在接続しているユーザーを返します。

使用例

➲ **使用例1**　現在のユーザーを返します：

```
SELECT CURRENT_USER;


          CURRENT_USER
==============================
SYSADM


1 rows selected
```

➲ **使用例2**　表sql99t2のuserカラムに現在接続しているユーザーを挿入し、そのデータを取り出します：

```
INSERT INTO sql99t2 VALUES (CURRENT_USER);
1 row inserted


]
```

関連関数

SESSION_USER

USER

4.30 CURTIME

目的

現在の時刻を返します。

構文

戻り値　　Time：現在の時刻

説明

現在の時刻を返します。

使用例

➲ **使用例**　現在の時刻を返します：

```
CURTIME()
```

関連関数

CURDATE

CURRENT_DATE

CURRENT_TIME

CURRENT_TIMESTAMP

NOW

4.31 DATABASE

目的

現在のセッションに接続されているデータベースの名前を返します。

構文

戻り値　　String： 接続されているデータベースの名前

説明

現在接続されているデータベースの名前を返します。別の方法としては、ODBCプログラムで**SQL_CURRENT_QUALIFIER**接続オプションを付けて**SQLGetConnectOption**を呼び出してデータベース名を取得することができます。

使用例

➲ **使用例**　現在接続されているデータベースの名前を返します：

```
DATABASE()
```

関連関数

USER

CURRENT_USER

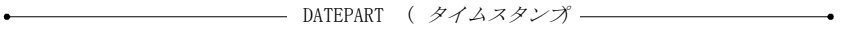
SESSION_USER

4.32 DATEPART

目的

タイムスタンプの中の日付を返します。

構文



タイムスタンプ	Timestamp： 日付を取り出すタイムスタンプ
戻り値	Date：*タイムスタンプ*の中の日付

説明

タイムスタンプの中の日付を返します。

使用例

➲ **使用例**　日付部分**1999-08-07**を返します：

```
DATEPART('1999-08-07 10:11:12.123')
```

関連関数

TIMEPART

4.33 DAYNAME

目的

日付の曜日名を返します。

構文

DAYNAME （ *日付* ）

日付　　Date：曜日名を求める日付

戻り値　　String： *日付* の曜日名

説明

日付の曜日の名前をデータソース固有の文字列（例、Sunday、Monday、…、Saturday）で返します。

使用例

➲ **使用例**　土曜日の文字列“**Saturday**”を返します：

```
DAYNAME('1999-12-25')
```

関連関数

DAYOFWEEK

MONTHNAME

4.34 DAYOFMONTH

目的

日付の中の日を整数で返します。

構文

日付	Date：日を求める日付
戻り値	Integer：*日付*の中の日

説明

日付の中の日を整数値 (1-31) で返します。

使用例

➲ **使用例**　日付の中の日の整数値**23**を返します：

```
DAYOFMONTH('1999-01-23')
```

関連関数

DAYOFWEEK

DAYOFYEAR

4.35 DAYOFWEEK

目的

日付の曜日を整数で返します。

構文

DAYOFWEEK （ *日付* ）

日付	Date：曜日を求める日付
戻り値	Integer： *日付* の曜日

説明

日付の曜日を整数で返します。曜日の整数は、1が日曜日、2が月曜日、…、7が土曜日です。

使用例

➲ 使用例1　水曜日の整数値3を返します：

```
DAYOFWEEK('2000-02-29')
```

➲ 使用例2　金曜日の整数値6を返します：

```
DAYOFWEEK('2000-03-03')
```

関連関数

DAYOFMONTH

DAYOFYEAR

DAYNAME

4.36 DAYOFYEAR

目的

日付に対応する年初からの日数を返します。

構文

DAYOFYEAR （ *日付* ）

日付	Date：年初から日数を求める日付
戻り値	Integer：日付 の年初からの日数

説明

日付に対応する年初からの日数を整数値1-366で返します。366は閏年の最後の日のときだけ返されます。

使用例

➲ **使用例1**　整数値**31**を返します：

```
DAYOFYEAR('1999-01-31')
```

➲ **使用例2**　整数値**365**を返します：

```
DAYOFYEAR('1999-12-31')
```

関連関数

DAYOFMONTH

DAYOFWEEK

QUARTER

WEEK

4.37 DAYS_BETWEEN

目的

二つの日付の間の日数を返します。

構文

日付1	Date：日数を計算する二つの日付の第一の日付
日付2	Date：日数を計算する二つの日付の第二の日付
戻り値	Integer：*日付1* と *日付2* の間の日数

説明

二つの日付の間の日数を返します。*日付1*は*日付2*の前でも後でもかまいません。

使用例

➲ **使用例1**　日付の間の日数**31**を返します：

```
DAYS_BETWEEN('1999-01-15', '1999-02-15')
```

➲ **使用例2**　日付の間の日数**31**を返します：

```
DAYS_BETWEEN('1999-02-15', '1999-01-15')
```

関連関数

SECS_BETWEEN

TIMESTAMPDIFF

4.38 DEGREES

目的

角度をラジアンから度に変換します。

構文

DEGREES （ *ラジアン* ）

ラジアン	Double：度に変換するラジアンの数値
戻り値	Double：ラジアン の度数

説明

ラジアンの角度を度に変換し倍精度浮動小数点数で返します。

使用例

➲ **使用例**　180度の近似値**1.79908747671078e+002**を返します：

```
DEGREES(3.14)
```

関連関数

RADIANS

4.39 DOCTOTXT

目的

Wordドキュメントを一時的なBLOB に変換します。

構文

Blob:　テキストに変換するカラム名
戻り値: blobがテキストに変換可能であれば一時BLOBはNCLOBで返されます。

説明

DOCTOTXT関数はMicrosoft Wordドキュメントを UnicodeのBlobテキストを含む一時BLOBに変換します。一時BlobまたはNULLが返されます。
DBMaster5.3のUDF は office 2007- 2010 バージョンでサポートできます。

使用例

➲ **使用例**　”**memo**”カラムをテキストに変換します

```
DOCTOTXT(memo)
```

4.40 EXISTSNODE

目的

指定ノードの有無のチェックに使われます。

構文

EXISTSNODE(XMLdata, xpath-expression, namespaces)

Xmldata……問い合わせるxml 内容

xpath-expression……xmldata問い合わせに使用します

Namespaces…….... xpath-expressionに使用されるnamespaceをオプションで指定します

Returnvalue……NCLOBで結果を順番に並べる

説明

existsNode関数は指定したノードが存在するかをチェックします。

使用例

➲ **使用例** **existsnode XML UDF**を使用して索引を作成します:

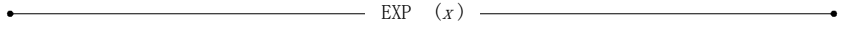
```
create index idx1 on t1 (existsnode(c1, '/order/items/item/@product', NULL));
```

4.41 EXP

目的

x の指数 e^x の値を返します。

構文



x	Double：自然対数 (e) のべき乗にする数値
戻り値	Double：自然対数 (e) の *x* 乗の値

説明

x の指数 e^x の値を倍精度浮動小数点数で返します。

使用例

➲ **使用例**　e^1 = e の値 2.71828182845905e+000を返します：

```
EXP(1)
```

関連関数

LOG

LOG10

POW

4.42 EXTRACT

目的

EXTRACT関数は複数、単一、ゼロの値を返します。

構文

戻り値　UDF: UDF結果の値は複数、単一、ゼロになります。

説明

複数、単一、ゼロの値を返します。asc/desc と一意索引を使用できません。

使用例

➲ 使用例　**extract XML UDF**を使用して索引を作成します:

```
create index idx1 on t1 (extract(c1, '/order/items/item/@product', NULL));
```

関連関数

LOG

LOG10

POW

4.43 EXTRACTVALUE

目的

単一、ゼロの値を返します。asc/descと一意索引が使用可能です。

構文

戻り値: UDF:　*単一、ゼロのUDF結果の値になります。複数の値にはなりません。*

説明

単一、ゼロの値を返します。asc/desc と一意索引を使用可能です。

使用例

➲ 使用例　**extractValue XML UDF**を使用して索引を作成します:

```
create index idx2 on t1 (extractValue(c1, '/order/items/item/@product',
NULL));
```

4.44 FILEEXIST

目的

ファイル・オブジェクトが物理的に存在するかどうかをチェックします。

構文

ファイル･オブジェクト File：存在をチェックするファイル・オブジェクト

戻り値 Integer：ファイルの存在、非存在を示す論理値

説明

ファイル・オブジェクトが物理的に存在するかどうかをチェックします。ファイルが存在する場合は1、存在しない場合は0が返されます。

使用例

➲ 使用例1　ファイルが存在することを示す1を返します：

```
FILEEXIST(file_column)
```

➲ 使用例2　ファイルが存在しないことを示す0を返します：

```
FILEEXIST(nofile_column)
```

関連関数

FILENAME

FILELEN

4.45 FILELEN

目的

ファイル・オブジェクトのファイル・サイズを返します。

構文

FILELEN （*ファイルオブジェクト*）

ファイル･オブジェクト File：サイズを求めるファイル・オブジェクト

戻り値 Integer：ファイルのバイト数

説明

ファイル・オブジェクトのサイズを返します。最大で2G-1、2Gを超える場合も同様の結果になります。*ファイル・オブジェクト*の引数は、FILEデータ型のカラムでなければなりません。

使用例

➲ **使用例** ファイルのサイズが**211**バイトとすると、整数**211**を返します：

```
FILELEN(file_column)
```

関連関数

FILEEXIST

FILENAME

4.46 FILELENEX

目的

*ファイルオブジェクト*のサイズを十進数の値として返します。

構文

FILELENEX (*fileobject*)

ファイルオブジェクト File:長さを表示するファイル

戻り値 Decimal:バイト単位でファイル長

説明

FILELENEX関数は*ファイルオブジェクト*のサイズを十進数の値として返します。*fileobject* 引数はデータベース内のFILE型のカラムでなければなりません。FOs ≥2Gの場合でも正しいサイズが返されます。

使用例

➲ 使用例　ファイルのサイズが211バイトとすると、整数211を返します

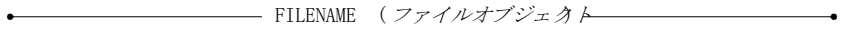
```
FILELENEX(file_column)
```

4.47 FILENAME

目的

ファイル・オブジェクトのファイル名を返します。

構文



ファイル･オブジェクト File：ファイル名を求めるファイル・オブジェクト

戻り値 String：ファイル名

説明

ファイル・オブジェクトの名前を文字列で返します。ファイル・オブジェクトの引数は、FILEデータ型のカラムでなければなりません。

使用例

➲ **使用例**　例えばファイル名**C：\PATH\MYFILE.FIL**を返します：

```
FILENAME(file_column)
```

関連関数

FILEEXIST

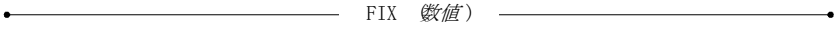
FILELEN

4.48 FIX

目的

実数の整数部分を返します。

構文



数値	Double：整数部分を求める実数
戻り値	Bigint：*数値* の整数部分

説明

実数の整数部分を整数値で返します。

使用例

➲ **使用例1**　整数**11**を返します：

```
FIX(11.99)
```

➲ **使用例2**　整数**12**を返します：

```
FIX(12.01)
```

➲ **使用例3**　整数**–11**を返します：

```
FIX(-11.99)
```

➲ **使用例4**　整数**–12**を返します：

```
FIX(-12.01)
```

関連関数

CEILING

FLOOR

MODFI

MODFM

RND

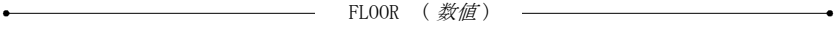
ROUND

4.49 FLOOR

目的

指定した数値より小さいか等しい直近の整数を返します。

構文



数値	Double：直近の床整数を求める数値
戻り値	Double：*数値*以下の直近の整数

説明

数値より小さいか等しい直近の整数を倍精度浮動小数点数で返します。

使用例

➲ **使用例1**　直近の整数値**1.20000000000000e+001**を返します：

```
FLOOR(12.01)
```

➲ **使用例2**　直近の整数値**1.10000000000000e+001**を返します：

```
FLOOR(11.99)
```

➲ 使用例3　直近の整数値**-1.20000000000000e+001**を返します：

```
FLOOR(-11.99)
```

➲ **使用例4**　直近の整数値**-1.30000000000000e+001**を返します：

```
FLOOR(-12.01)
```

関連関数

CEILING

FIX

MODFI

MODFM

RND

ROUND

4.50 FREXPE

目的

数値に対して、*数値* = $X \times 2^n$ を満たす *n* を返します。

構文

FREXPE （*数値*）

数値	Double： *数値* = $X \times 2^n$ を満たす *n* を求める数値
戻り値	Integer： *数値* = $X \times 2^n$ の等式を満たす *n*

説明

数値 = X × 2^n の等式を満たす整数 *n* を返します。Xは0.5=<｜X｜ =<1の範囲の値です。

使用例

➲**使用例**　整数**3**を返します。Xが0.5 ≤｜X｜ ≤1の範囲内で 4.0 = X × 2^n を満たす***n***は**3**でなければならないことを意味します：

```
FREXPE(4.0)
```

関連関数

FREXPM

LDEXP

4.51 FREXPM

目的

数値に対して、*数値* = X × 2^n を満たすXを返します。

構文

FREXPM （ *数値* ）

数値	Double： *数値* = X × 2^n を満たすXを求める数値
戻り値	Double： *数値* = X × 2^n の等式を満たすX

説明

数値 = X × 2^n の等式を満たすXを返します。Xは0.5 ≤ | X| ≤ 1の範囲の値です。

使用例

➲ **使用例1　5.00000000000000e-001**を返します；*数値* が 4.0 で *n* が整数のときは、Xは0.5つまり **5.00000000000000e-001** でなければなりません：

```
FREXPM(4.0)
```

関連関数

FREXPE

LDEXP

4.52 FTOA

目的

指定した数値と小数点以下の桁数をもつ数値文字列を返します。

構文

FTOA （*数値, 桁数, フォーマット*）

数値	Double：文字列に変換する数値
桁数	Integer： 小数点以下の桁数
フォーマット	String： 戻り値のフォーマット
戻り値	String：指定した小数点以下の桁数とフォーマットの数値文字列

説明

指定した小数点以下の桁数をもつ数値の文字列を返します。*桁数* は小数点以下の桁数を指定し、*フォーマット* は通常の10進数フォーマットか指数フォーマットかを指定します。

フォーマットは“f”、“F”、“e”、“E”で指定します。“f” と “F” は10進数フォーマット、例えば桁数を2とすると123.45のような文字列を返します。“e”と“E”は指数フォーマット、例えば1.23e+02のような文字列を返します。指数フォーマットの数値は、10進数フォーマットと同じ数値に変換されます。

使用例

➲ **使用例1**　文字列"**123.46**"を返します：

```
FTOA(123.456789, 2, 'f')
```

➲ **使用例2**　文字列"**1.23e+02**"を返します：

```
FTOA(123.456789, 2, 'e')
```

関連関数

ATOF

4.53 HIGHLIGHT

目的

ソース内で探索したテキストの前後にタグを付けて強調します。

構文

HIGHLIGHT （*テキスト, パターン, 感度, 前タグ, 後タグ*

テキスト	Long varchar：ソーステキスト
パターン	Char：強調するテキストのパターンまたはパターン論理式
感度	Integer：大文字小文字の区別、1 は区別し 0 は区別しない
前タグ	Char：パターンの前に付けるタグ、NULL はタグを付けないことを示す
後タグ	Char：パターンの後に付けるタグ、NULL はタグを付けないことを示す
戻り値	BLOB：強調パターンで変更されたテキスト

注：　最大10000 (MaxTagSpace) バイトまでタグを付けることができます。論理演算子 [&, \|, !, (,)] がパターンの中にあるときは、全ての個々のパターンにタグが付けられます。ただし、！(NOT) パターンは付けられません。入力テキストは、long varchar、file、char タイプにすることができます。

説明

パターンにマッチした全てのテキストの前後に前タグ／後タグを付けて強調し、変更されたソーステキストを返します。

使用例

➲ **使用例1　contentの全ての“Intel”と“AMD”に前タグ“<<”と後タグ“>>”を付けて強調したテキストを返します：**

```
SELECT highlight(content,'Intel | AMD',0,'<<','>>') FROM news WHERE content
MATCH 'Intel| AMD';
```

関連関数

HTMLHIGHLIGHT

HITCOUNT

HITPOS

HTMLTITLE

BLOBLEN

4.54 HITCOUNT

目的

ソース・テキスト内に見つかったパターンの個数をカウントします。

構文

HITCOUNT （*テキスト*, *パターン*, *感度*）

テキスト	Long varchar：ソース・テキスト
パターン	Char：個数をカウントするパターンまたはパターン論理式

感度	Integer：大文字小文字の区別、1は区別し0は区別しない
戻り値	Integer：ソース・テキスト内のパターンの個数

注：　パターン論理式のカウント規則：

a AND b ： min(count(a), count(b))

a OR b ： count(a) + count(b)

NOT a ： count = 0

説明

ソース・テキスト内のパターンの個数を返します。

使用例

➲ **使用例**　ソースデータ“**content**”内で見つかった“**target**”の個数を返します。大文字と小文字は区別しません：

```
HITCOUNT(content, “target”, 0)
```

関連関数

HIGHLIGHT

HTIPOS

HTMLTITLE

HTMLHIGHLIGHT

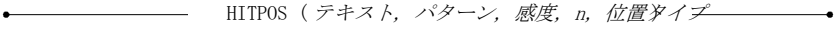
BLOBLEN

4.55 HITPOS

目的

ソース・テキストでn-番目にマッチしたパターンの位置情報を次のいずれかで返します。先頭からのオフセット、末尾からのオフセット、テキスト長、バイナリオフセット（先頭からは上位24ビット、末尾からは下位8ビット）

構文



テキスト	Long varchar：ソース・テキスト
パターン	Char：位置情報を取得するパターンまたはパターン論理式
感度	Integer：大文字小文字の区別、1 は区別し 0 は区別しない
n	Integer：ソーステキスト内の n-番目のパターン
位置タイプ	Char：以下の位置タイプ： 0：先頭からのオフセット (初期設定の設定) 1：末尾からのオフセット 2：パターンの長さ 3：バイナリオフセット (先頭からは上位 24 ビット、末尾からは下位 8 ビット)
戻り値	Integer：ソーステキスト内のn^{th}番目のパターンの位置情報。n^{th}番目のパターンがないときは 0

注：　オフセットは1 から始まります。

説明

ソース・テキスト内の *n-番目* のパターンの位置情報を返します。

使用例

➲ **使用例**　ソース・テキストを“a b A c” とすると、以下の例は5、3、5、7を返します：

```
HITPOS(src,'A',   1, 1, 0) = 5  ('A')
HITPOS(src,'A&B' 0, 2, 0) = 3  ('b')
HITPOS(src,'a|b|c', 0, 3, 0) = 5  ('A')
HITPOS(src,'!a&c' 0, 1, 0) = 7  ('c')
```

関連関数

HIGHLIGHT

HTICOUNT

HTMLTITLE

HTMLHIGHLIGHT

BLOBLEN

4.56 HMS

目的

指定した時・分・秒の時刻を返します。

構文

HMS （*時, 分秒*）

時	Integer：時刻の時成分
分	Integer：時刻の分成分
秒	Integer：時刻の秒成分
戻り値	Time：時・分・秒の時刻

説明

Timeフォーマットの時刻を返します。時は0～23で与えます。時は24時間形式です。12時間形式のAMおよびPMの時刻を与えることはできません。分は0～59で与えます。秒は0～59で与えます。

使用例

➲ **使用例1**　時刻**10:11:12**を返します。**10:11:12AM**の時刻です：

```
HMS(10, 11, 12)
```

➲ **使用例2**　時刻**22:11:12**を返します。**10:11:12PM**の時刻です：

```
HMS(22, 11, 12)
```

関連関数

MDY

HOUR

MINUTE

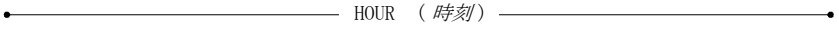
SECOND

4.57 HOUR

目的

時刻の中の時部分を返します。

構文



時刻	Time：時部分を求める時刻
戻り値	Integer：*時刻*の中の時

説明

時刻の中の時を24時間形式の整数0～23で返します。

使用例

➲ **使用例1**　10時の整数**10**を返します：

```
HOUR('10:11:12')
```

➲ **使用例2**　22時の整数**22**を返します：

```
HOUR('PM 10:11:12')
```

関連関数

MINUTE

SECOND

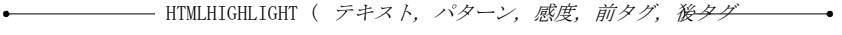
HMS

4.58 HTMLHIGHLIGHT

目的

HTMLファイル内のパターンを引用形にして強調します。HTML文書構造は壊しません。

構文



テキスト	Long varchar：ソース・テキスト
パターン	Char：強調するパターンまたはパターン論理式
感度	Integer：大文字小文字の区別、1 は区別し 0 は区別しない
前タグ	Char：パターンの前に付けるタグ、NULLはタグを付けないことを示す
後タグ	Char：パターンの後に付けるタグ、NULLはタグを付けないことを示す
戻り値	BLOB：強調パターンで変更されたテキスト

*注：　最大10000 (MaxTagSpace) バイトまでタグを付けることができます。論理演算子 [&, |, !, (,)] がパターンの中にあるときは、全ての個々のパターンにタグが付けられます。ただし、！(NOT) パターンには付けられません。入力テキストは、long varchar、file、charタイプにすることができます。HTMLタグ内の内容 (コメントを含む) は強調しません。全てのHTMLタグ (コメントを含む) は空白とみなされます。例えばパターンを “DBMaster License” とすると、HTML データ “DBMaster
License” は強調されます。しかし HTML データ “<B>DBM</B>aker” は “DBMaster” パターンにマッチしません。<BODY> 以降のデータだけが強調されます。*

説明

マッチした全てのテキスト・パターンの前後に前タグと後タグを付けて強調し、変更されたソース・テキストを返します。

使用例

➲ **使用例**　**content**の全ての“**Intel**”と“**AMD**”に前タグ**“<<”**と後タグ**“>>”**を付けて強調したテキストを返します：

```
HTMLHIGHLIGHT(content、'Intel | AMD',0,'<<','>>')
```

関連関数

HIGHLIGHT

HITCOUNT

HITPOS

HTMLTITLE

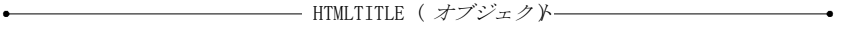
BLOBLEN

4.59 HTMLTITLE

目的

ソースHTMLデータのタイトル (HTMLタグ“*<title>*”と“*</title>*”の間にあるテキスト) を見つけます。

構文



オブジェクト	Long varbinary：ソースHTMLデータ
戻り値	Varchar：ソースHTMLデータのタイトル

説明

ソースHTMLデータのタイトル・テキストを返します。

使用例

➲ **使用例**　ソースHTMLデータ“**htmlFile**”のタイトル文字列を返します：

```
HTMLTITLE(htmlFile)
```

関連関数

HIGHLIGHT

HITCOUNT

HTIPOS

HTMLHIGHLIGHT

BLOBLEN

4.60 HTMTOTXT

目的

htmlドキュメントを一時的なBLOB に変換します。

構文

Blob:　テキストに変換するカラム名
戻り値: BLOBがテキストに変換可能であればCLOB型の一時BLOB

説明

HTMTOTXT関数はhtmlドキュメントをBLOBローカルコードを含む一時BLOBへの変換に使用します。

使用例

➲ **使用例:**　カラム"memo"をテキストに変換します。

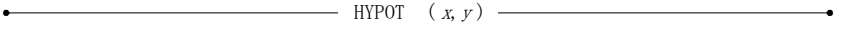
```
HTMTOTXT(memo)
```

4.61 HYPOT

目的

直角三角形の斜辺の長さを返します。

構文



x	Double：直角三角形の底辺の長さ
y	Double：直角三角形の高さ
戻り値	Double：直角三角形の斜辺の長さ

説明

直角三角形の斜辺の長さを倍精度浮動小数点数で返します。直角三角形の斜辺の長さzは、$z^2=x^2+y^2$ (ピタゴラスの定理) から計算します。

使用例

➲ **使用例**　直角三角形の斜辺の長さ**5**を返します：

```
HYPOT(3,4)
```

4.62 INSERT

目的

文字列の中の部分文字列を別の文字列で置き換えます。文字列を挿入するだけにすることもできます。

構文

INSERT （*文字列1*, *開始位置*, *長さ*, *文字列2*）

文字列1	String：挿入または置き換えの対象文字列
開始位置	Integer：*文字列1*の挿入位置
長さ	Integer：*文字列1*の置き換える文字列の長さ
文字列2	String：*文字列1*に挿入する文字列
戻り値	String：*文字列1*に*文字列2*を置き換え・挿入した文字列

説明

INSERT関数は、*文字列1*の*開始位置*から*長さ*の文字を*文字列2*によって置き換えた文字列を返します。*文字列1*の*開始位置* は*文字列2*の先頭文字が置かれる位置です。*長さ*を0にすると、置き換えずに*文字列2*を挿入します。全ての引数が与えられないとエラーになります。

文字列が NULL 値の場合または数値が異常の場合、関数が返す文字列は以下の規則で決められます：

- 文字列1が NULL のときは、NULL を返します。
- 開始位置、長さ、文字列2のどれかがNULLのときは、文字列1を返します。
- 開始位置、長さのどれかが0以下のときは、文字列1を返します。
- 開始位置が 文字列1の長さより大きいときは、文字列1を返します。

使用例

➲ **使用例1**　文字列**“Good ng!”**を返します：

```
INSERT(‘morning!’, 1, 5, ‘Good ’)
```

➲ **使用例2**　文字列**“Good morning!”**を返します：

```
INSERT(‘Good ’, 6, 8, ‘morning!’)
```

➲ **使用例3**　文字列**“Good night!”**を返します：

```
INSERT(‘Good morning!’, 6, 7, ‘night’)
```

➲ **使用例4**　文字列**“Good morning, sir. Here is your coffee.”**を返します：

```
INSERT(‘Good morning! Here is your coffee.’, 13, 1, ‘, sir.’)
```

関連関数

REPLACE

4.63 INVDATE

目的

正しい日付かどうかをチェックします。

構文

INVDATE （*日付*）

日付	Date： 正しさをチェックする日付
戻り値	Integer：日付が正しいかどうかを示す整数

説明

引数の*日付*が正しいかどうかをチェックします。戻り値は以下のようになります：

- 1 不正な日付 (例、 日付の範囲外)
- 0 正しい日付 (例、 ‘0001-01-01’ ～ ‘9999-12-31’)
- -1不定の日付 (例、 NULL 値)

使用例

➲ **使用例**　正しい日付なので**0**を返します：

```
INVDATE(‘2000-01-01’)
```

関連関数

INVTIME

INVTIMESTAMP

4.64 INVTIME

目的

正しい時刻かどうかをチェックします。

構文

INVTIME （*時刻*）

時刻	Time：正しさをチェックする時刻
戻り値	Integer：時刻が正しいかどうかを示す整数

説明

引数の*時刻*が正しいかどうかをチェックします。戻り値は以下のようになります：

- 1 不正な時刻 (例、 時刻の範囲外)
- 0 正しい時刻 (例、 ‘00：00：00’ ～ ‘24：00：00’)
- -1不定の時刻 (例、 NULL 値)

使用例

➲ **使用例**　正しい時刻なので **0** を返します：

```
INVTIME('01:01:01')
```

関連関数

INVDATE

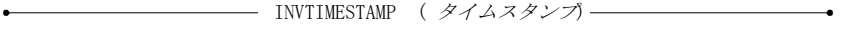
INVTIMESTAMP

4.65 INVTIMESTAMP

目的

正しいタイムスタンプかどうかをチェックします。

構文



タイムスタンプ	Timestamp：正しさをチェックするタイムスタンプ
戻り値	Integer：タイムスタンプが正しいかどうかを示す整数

説明

引数のタイムスタンプが正しいかどうかをチェックします。戻り値は以下のようになります：

- 1 不正なタイムスタンプ (例、 タイムスタンプの範囲外)
- 0 正しいタイムスタンプ (例、 ‘00：00：00’ ～ ‘24：00：00’)
- -1不定のタイムスタンプ (例、 NULL 値)

使用例

➲ 使用例　正しいタイムスタンプなので 0 を返します：

```
INVTIMESTAMP('1999-08-07 10:11:12.123' )
```

関連関数

INVTIME

INVDATE

4.66 LAST_DAY

目的

日付の月末の日付を返します。

構文

日付	Date：月末の日付を求める日付
戻り値	Date：指定した*日付*の月末の日付

説明

指定した*日付*の月末の日付を返します。

使用例

➲ **使用例1**　2月の月末日‘**1996-02-29**’を返します：

```
LAST_DAY('1996-02-08')
```

➲ **使用例2**　12月の月末日‘**2002-12-31**’を返します：

```
LAST_DAY('2002-12-25')
```

関連関数

NEXT_DAY

4.67 LCASE

目的

小文字の文字列に変換します。

構文

LCASE （*文字列*）

文字列	String：小文字に変換するテキスト
戻り値	String：*文字列*を小文字に変換したテキスト

説明

LCASE関数は、全ての大文字を小文字に変換します。数値や記号は変換されません。*文字列*がNULL のときは、NULL値が返されます。*文字列*の引数を与えないとエラーになります。

使用例

➲ **使用例1**　文字列“**abcdef**”を返します：

```
LCASE('ABCdef')
```

➲ **使用例2**　文字列“**abc123**”を返します：

```
LCASE('ABC123')
```

➲ **使用例3**　文字列“**abc@#$**”を返します：

```
LCASE('ABC@#$')
```

関連関数

LOWER

UCASE

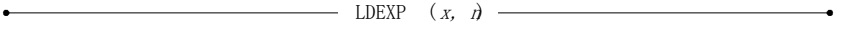
UPPER

4.68 LDEXP

目的

x と n から $x \times 2^n$ を計算します。

構文



x	Double： $x \times 2^n$ のマンティッサ *x*
n	Integer： $x \times 2^n$ の指数 *n*
戻り値	Double： $x \times 2^n$ の結果の数値

説明

$x \times 2^n$ の結果の数値を倍精度浮動小数点数で返します。

使用例

➲**使用例**　$8 = 0.5 \times 2^4$ の結果を **8.00000000000000e+000** で返します：

```
LDEXP(0.5, 4)
```

関連関数

FREXPE

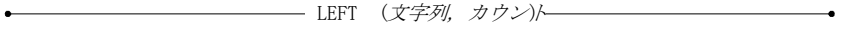
FREXPM

4.69 LEFT

目的

文字列の左 (先頭) から取り出した文字列を返します。

構文



文字列	String：文字列を取り出す文字列
カウント	Integer：取り出す文字数
戻り値	String：*文字列* の左から取り出した*カウント* 数の文字列

説明

LEFT関数は、文字列の先頭から指定カウントの文字列を取り出して返します。カウントが0以下のときはNULL値が返されます。全ての引数が指定されていないとエラーになります。

使用例

➲ **使用例**　4文字の文字列“**Good**”を返します：

```
LEFT('Good morning!', 4)
```

関連関数

LTRIM

RIGHT

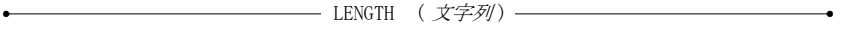
SUBSTRING

4.70 LENGTH

目的

文字列の長さ (文字数) を返します。

構文



文字列	String：長さを求める文字列
戻り値	Integer：*文字列*の長さ (文字数)

説明

LENGTH関数は文字列の長さを返します。文字列の後ろの空白文字や終了文字は長さに含まれません。文字列の引数が無いとエラーになります。

使用例

➲ **使用例**　文字列の長さ**13**を返します：

```
LENGTH('Good morning!   ')
```

関連関数

CHAR_LENGTH

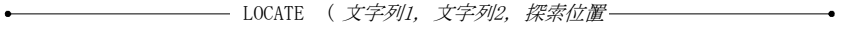
CHARACTER_LENGTH

4.71 LOCATE

目的

文字列の中にある文字列を探索し見つけた位置を返します。

構文



文字列1	String：探索する文字列
文字列2	String：探索の対象になる文字列
探索位置	Integer：*文字列2*の探索開始位置
戻り値	Integer：*文字列2*の中の*文字列1*の位置

説明

LOCATE関数は、*文字列2*の中にある*文字列1*の最初の出現の開始位置を返します。指定した*探索位置*から*文字列1*の最初の出現を探索します。*探索位置*を1にすると、*文字列2*の最初から探索します。*文字列1*が*文字列2*に無いときは0が返されます。文字列がNULL値の場合または探索位置が異常の場合、関数が返す値は以下の規則で決められます：

- 文字列1がNULLのときは、NULLを返します。
- 文字列2または探索位置がNULLのときは、0を返します。
- 探索位置が0以下のときは、1桁目から探索します。
- 探索位置が文字列2の長さ＋1より大きいときは、0を返します。

使用例

➲ **使用例1**　**4**を返します：

```
LOCATE('def', 'abcdefghi', 1)
```

➲ **使用例2**　**0**を返します：

```
LOCATE('def', 'abcdefghi', 5)
```

➲ **使用例3　4**を返します：

```
LOCATE('def', 'abcdefghi', 4)
```

➲ **使用例4　4**を返します：

```
LOCATE('def', 'abcdefghi', -1)
```

➲ **使用例5　0**を返します：

```
LOCATE('def', 'abcdefghi', 10)
```

関連関数

POSITION

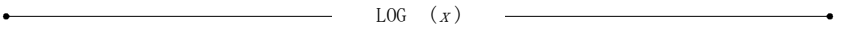
SUBSTRING

4.72 LOG

目的

x の自然対数を計算します。

構文



x	Double：自然対数を求める値
戻り値	Double：「*x*」の自然対数

説明

x の自然対数を倍精度浮動小数点数で返します。

使用例

➲ **使用例　1.00000000000000e+000** を返します：

```
LOG(2.71828182845905e+000)
```

関連関数

EXP

LOG10

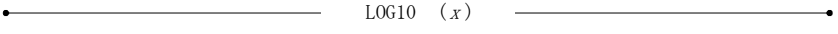
POW

4.73 LOG10

目的

*x*の10を底とする対数を計算します。

構文



x	Double：10を底とする対数を求める数値
戻り値	Double：10を底とする*x*の対数

説明

*x*の10を底とする対数を倍精度浮動小数点数で返します。

使用例

➲ **使用例　2**を返します：

```
LOG10(100)
```

関連関数

EXP

LOG

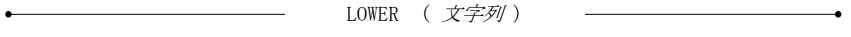
POW

4.74 LOWER

目的

小文字の文字列に変換します。

構文



文字列	String：小文字に変換するテキスト
戻り値	String：*文字列*を小文字に変換したテキスト

説明

LCASEと同様の働きをします。小文字の文字列に変換します。

使用例

➲ 使用例　：

```
SELECT LOWER('ABCDEF');
LOWER('ABCDEF')
===============
abcdef
```

関連関数

LCASE

UCASE

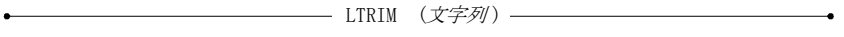
UPPER

4.75 LTRIM

目的

文字列の先頭にある空白文字を削除します。

構文



文字列	String：先頭にある空白文字を削除する文字列
戻り値	String：空白文字を削除した文字列

説明

LTRIM関数は、文字列の引数の先頭にある空白文字を削除した文字列を返します。引数を指定しないとエラーになります。

使用例

➲ **使用例**　文字列“**Good morning!**”を返します：

```
LTRIM('   Good morning!')
```

関連関数

LEFT

RTRIM

SUBSTRING

TRIM

4.76 MDY

目的

月・日・年の日付を返します。

構文

月	Integer：日付の月
日	Integer：日付の日
年	Integer：日付の年
戻り値	Date：*月*、*日*、*年*から構成した日付フォーマットの日付

説明

月・日・年で指定した日付を現在の日付フォーマットで返します。月は0～12で与えます。日は0～31で与えます。年は0001～9999で与えます。

使用例

➲ **使用例1**　現在の日付フォーマットを**yyyy-mm-dd**と設定していると、**1996-02-08**を返します：

```
MDY(2,8,1996)
```

➲ **使用例2**　現在の日付フォーマットを**mm/dd/yyyy**と設定していると、**02/08/2001**を返します：

```
MDY(2,8,2001)
```

関連関数

HMS

DAYOFMONTH

MONTH

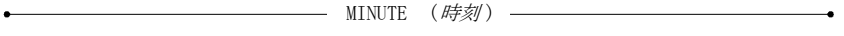
YEAR

4.77 MINUTE

目的

時刻の中の分を返します。

構文



時刻	Time：分を求める時刻
戻り値	Integer：*時刻*の中の分

説明

時刻の中の分を0～59の整数で返します。

使用例

➲ **使用例**　分の整数**11**を返します：

```
MINUTE('10:11:12')
```

関連関数

HOUR

SECOND

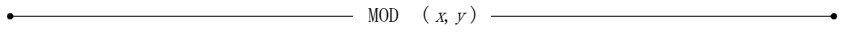
HMS

4.78 MOD

目的

割り算の余りを計算します。

構文



x	Double：被除数
y	Double：除数
戻り値	Double：除算の余り

説明

*x*を*y*で割り算した余りを倍精度浮動小数点数で返します。

使用例

➲ **使用例**　割り算の余り**2.00000000000000e+000**を返します：

```
MOD(17, 3)
```

4.79 MODFI

目的

実数の整数部分を返します。

構文

MODFI （*数値*）

数値	Double：整数部分を求める実数
戻り値	Double：*数値* の整数部分

説明

実数の整数部分を倍精度浮動小数点数で返します。

使用例

➲ **使用例1**　整数部分**3.00000000000000e+000**を返します：

```
MODFI(3.1415926535897936)
```

➲ **使用例2**　整数部分**-3.00000000000000e+000**を返します：

```
MODFI(-3.1415926535897936)
```

関連関数

CEILING

FIX

FLOOR

MODFM

RND

ROUND

4.80 MODFM

目的

実数の小数部分を返します。

構文

数値	Double：小数部分を求める実数
戻り値	Double：*数値* の小数部分

説明

実数の小数部分を倍精度浮動小数点数で返します。

使用例

➲ **使用例1**　小数部分**1.41592653589790e-001**を返します：

```
MODFM(3.1415926535897936)
```

➲ **使用例2**　小数部分**-1.41592653589790e-001**を返します：

```
MODFM(-3.1415926535897936)
```

関連関数

CEILING

FIX

FLOOR

MODFI

RND

ROUND

4.81 MONTH

目的

日付の中の月を返します。

構文

日付	Date：月を求める日付
戻り値	Integer：*日付*の中の月

説明

日付の中の月を1～12の整数で返します。

使用例

➲ **使用例**　月の整数**2**を返します：

```
MONTH('1996-02-29')
```

関連関数

DAYOFMONTH

QUARTER

YEAR

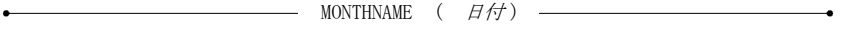
WEEK

4.82 MONTHNAME

目的

日付の月の名前を返します。

構文



日付	Date：月の名前を求める日付
戻り値	String：*日付*の月の名前

説明

日付の月の名前をデータソース固有の文字列（例、JAN、FEB、…、DEC）で返します。日付の引数が無効のときはエラーが返されます。

使用例

➲ **使用例1　2**月の名前“**FEB**”を返します：

```
MONTHNAME('1996-02-29')
```

関連関数

DAYNAME

4.83 NEXT_DAY

目的

指定した日付より後の次の指定した曜日の日付を返します。

構文

NEXT_DAY （*日付, 曜日*）

日付	Date：次の指定した曜日の日付を求める日付
曜日	String：日付を求める曜日
戻り値	Date：*日付*より後の次の*曜日*の日付

説明

指定した日付より後の次の指定した曜日の日付を返します。曜日の引数は、（Monday、Tuesday、 …、Sunday）または短縮形（Mon、Tue、…、Sun）で指定します。曜日の大文字と小文字は区別しません。

使用例

➲ **使用例1**　次の月曜日の日付**1996-03-04**を返します：

```
NEXT_DAY('1996-02-29', 'Monday')
```

➲ **使用例2**　次の火曜日の日付**1996-03-05**を返します：

```
NEXT_DAY('1996-02-29', 'Tuesday')
```

関連関数

LAST_DAY

4.84 NOW

目的

現在の日付と時刻のタイムスタンプを返します。

構文

戻り値　　Timestamp：現在の日付と時刻

説明

現在の日付と時刻をタイムスタンプの値として返します。

関連関数

CURDATE

CURTIME

CURRENT_DATE

CURRENT_TIME

CURRENT_TIMESTAMP

4.85 PDFTOTXT

目的

pdfドキュメントを一時的なBLOB に変換します。

構文

Blob: テキストに変換するカラム名

戻り値: BLOBが変換可能な場合、一時BLOBはNCLOBで表示されます

説明

PDFTOTXT関数はPDFドキュメントをUnicodeのテキストBLOBを含む一時BLOBへの変換に使用されます。一時BLOBまたはNULLを返します。DBMasterはPDFの1.2, 1.3, 1.4, 1.5, 1.6 と1.7でサポートすることを注意してください。

使用例

➲ **使用例** カラム"**memo**"をテキストに変換します

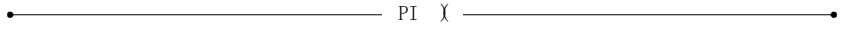
```
PDFTOTXT(memo)
```

4.86 PI

目的

πの定数を返します。

構文



戻り値 Decimal：π の値

説明

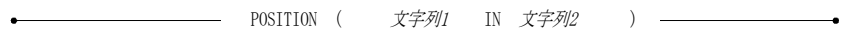
πの値3.1415926535897936をDecimal（精度17、スケール16）データ型で返します。

4.87 POSITION

目的

文字列2の中にある文字列 1 を探索し見つけた位置を返します。

構文



文字列1 String: 位置を見つける文字列

文字列2 String: 検索する文字列

戻り値 Integer: *文字列2*にある*文字列1*の開始位置

説明

「*文字列2*」にある「*文字列 1*」の最初の出現の開始位置を返します。「*文字列 1*」が「*文字列2*」にない場合、戻り値は0になります。文字列にNULL値が含まれるかどうかを判断するために以下のルールを用います。

- 文字列1にNULL 値がある場合、戻り値はNULL 値になります。
- 文字列2、或いは開始位置にNULL 値がある場合、戻り値は0になります。

使用例

➲ **使用例1** 次の**SQL**文は「**4**」を返します：

```
SELECT POSITION('abc' in 'defabcjlkjl');
POSITION('ABC' IN 'DEFABCJLKJL')
==============================
             4
```

➲ **使用例2** 次の**SQL**文は、「**1**」を返します：

```
select position('abc' in 'abcdefghihj');
POSITION('ABC' IN 'ABCDEFGHIHJ')
==============================
              1
```

➲ **使用例3** 次の**SQL**文は、「**0**」を返します：

```
select position('abc' in 'jlkjlkklj');
POSITION('ABC' IN 'JLKJLKKLJ')
============================
              0
```

関連関数

SUBSTRING

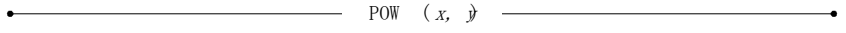
LOCATE

4.88 POW

目的

x^y の値を返します。

構文



x	Double：x^y の *x*
y	Double：x^y の *y*
戻り値	Double：x^y の値

説明

x^yの値を倍精度浮動小数点数で返します。

使用例

➲ **使用例　23**の値**8.00000000000000e+000**を返します：

```
POW(2, 3)
```

関連関数

EXP

LOG

LOG10

4.89 PPTTOTXT

目的

PowerPointドキュメントを一時的なBLOB に変換します。

構文

Blob: テキストに変換するカラム名

戻り値: テキストに変換可能な場合、一時BLOBはNCLOBで表示されます

説明

PPTTOTXT関数はMicrosoft PowerPointドキュメントをUnicodeのBlobのテキストを含む一時BLOBに変換します。一時BlobまたはNULLを返します。DBMaster5.3のUDFは office 2007- 2010 バージョンでサポートします。

使用例

➲ **例** "**memo**"カラムをテキストに変換します

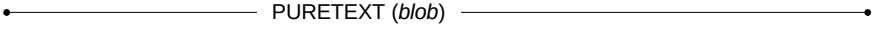
```
PPTTOTXT(memo)
```

4.90 PURETEXT

目的

blobを一時的なBLOB に変換します

構文



Blob: テキストに変換するカラム名

戻り値: テキストに変換可能な場合一時BLOBはNCLOBで表示されます

説明

PURETEXT関数はBLOBをUnicodeのBlobテキストを含む一時BLOBに変換します。

Media型カラム、text converter使用のドメインでPURETEXTを使用するときtext converter関数を呼び出します。

使用例

➲ **使用例　"memo"**カラムをテキストに変換します

```
PURETOTXT(memo)
```

4.91 QUARTER

目的

日付の四半期の整数を返します。

構文

QUARTER （ *日付* ）

日付	Date：四半期を求める日付
戻り値	Integer：*日付*の四半期の整数

説明

日付の四半期の整数1～4を返します。1は1月1日～3月31日の第一四半期を表します。

使用例

➲ **使用例**　第一四半期の**1**を返します：

```
QUARTER('2002-01-20')
```

関連関数

DAYOFMONTH

MONTH

WEEK

YEAR

4.92 RADIANS

目的

角度の度をラジアンに変換します。

構文

RADIANS （*度数*）

度数	Double：ラジアンに変換する度数
戻り値	Double：*度数*をラジアンに変換した数値

説明

度数を変換したラジアンの数値を倍精度浮動小数点数で返します。

使用例

➲ **使用例**　180度のラジアンの値3.14159265358979e+000を返します：

```
RADIANS(180)
```

関連関数

DEGREES

4.93 RAND

目的

乱数を返します。

構文

戻り値　　Integer：乱数

説明

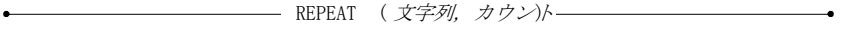
整数の乱数を返します。

4.94 REPEAT

目的

文字列を指定した回数だけ繰り返します。

構文



文字列　　String：繰り返される文字列

カウント　　Integer：繰り返す回数

戻り値　　String：*文字列* を*カウント*の回数繰り返した文字列

説明

REPEAT関数は、指定した文字列を指定した回数繰り返した文字列を返します。文字列の引数がNULLまたは空列のときは以下の規則に従います：

- 文字列または カウントが NULL のときは、 NULL が返されます。
- 文字列が空列であるかカウントが0以下のときは、空列が返されます。
- 全ての引数を指定しないとエラーになります。

使用例

➲ **使用例1**　文字列**“Good morning! Good morning! ”**を返します：

```
REPEAT('Good morning! ', 2)
```

➲ **使用例2**　文字列**“Zzzz Zzzz Zzzz Zzzz ”を返します：**

```
REPEAT('Zzzz ', 4)
```

関連関数

CONCAT

SPACE

4.95 REPLACE

目的

文字列内の指定した文字列の全ての出現を別の文字列で置き換えます。

構文

REPLACE （ *文字列1*, *文字列2*, *文字列3*

文字列1	String：置き換え対象の文字列
文字列2	String：置き換えられる文字列
文字列3	String：置き換える文字列

戻り値　　String：*文字列1*内の全ての*文字列2*の出現を*文字列3*で置き換えた文字列

説明

REPLACE関数は、*文字列1*内の全ての*文字列2*の出現を*文字列3*で置き換えます。文字列の引数がNULLまたは空列(長さが0)のときは以下の規則に従います：

- 文字列1がNULLのときは、NULLが返されます。
- 文字列2または文字列3がNULLのときは、文字列1が返されます。
- 文字列2が空列のときは、文字列1が返されます。

使用例

➲ **使用例1**　文字列**"Good evening! Good evening!"**を返します：

```
REPLACE('Good morning! Good morning!', 'morning', 'evening')
```

➲ **使用例2**　文字列**"Goodbye Dave."**を返します：

```
REPLACE('Hello, Dave.', 'Hello,', 'Goodbye')
```

関連関数

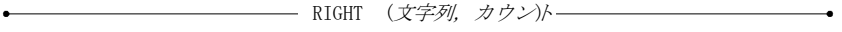
INSERT

4.96 RIGHT

目的

文字列の右側 (末尾) の文字列を返します。

構文



文字列	String：文字列を取り出す文字列
カウント	Integer：取り出す文字の個数
戻り値	String：*文字列* の右側から*カウント* 個の文字を取り出した文字列

説明

RIGHT関数は、文字列の末尾からカウント個の文字を取り出した文字列を返します。カウントが0以下のときは、NULL値が返されます。全ての引数を指定しないとエラーになります。

使用例

➲ **使用例**　文字列**"morning! "**を返します：

```
RIGHT('Good morning!  ', 10)
```

注：　使用例の関数引数と戻り値には、感嘆符 (!) の後に2個の空白があります。

関連関数

LEFT

RTRIM

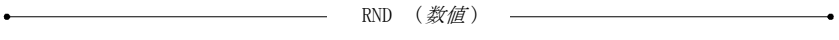
SUBSTRING

4.97 RND

目的

数値を四捨五入して整数に丸めます。

構文



数値	Double：四捨五入して丸める数値
戻り値	Bigint：「*数値*」を四捨五入して丸めた整数

説明

数値を四捨五入して整数に丸めます。

使用例

➲ **使用例1**　四捨五入して丸めた整数12を返します：

```
RND(12.01)
```

➲ **使用例2**　四捨五入して丸めた整数12を返します：

```
RND(12.49)
```

➲ **使用例3**　四捨五入して丸めた整数13を返します：

```
RND(12.50)
```

➲ **使用例4**　四捨五入して丸めた整数13を返します：

```
RND(12.99)
```

関連関数

CEILING

FIX

FLOOR

MODFI

MODFM

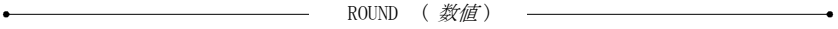
ROUND

4.98 ROUND

目的

数値を四捨五入して整数に丸めます。

構文



数値	Double：四捨五入して丸める数値
戻り値	Bigint：*数値* を四捨五入して丸めた整数

説明

数値を四捨五入して整数に丸めます。

使用例

➲ **使用例1**　四捨五入して丸めた整数**12**を返します：

```
ROUND(12.01)
```

➲ **使用例2**　四捨五入して丸めた整数**12**を返します：

```
ROUND(12.49)
```

➲ **使用例3**　四捨五入して丸めた整数**13**を返します：

```
ROUND(12.50)
```

➲ **使用例4**　四捨五入して丸めた整数**13**を返します：

```
ROUND(12.99)
```

関連関数

CEILING

FIX

FLOOR

MODFI

MODFM

RND

4.99 RTRIM

目的

文字列の末尾の空白を削除します。

構文

RTRIM （*文字列*）

文字列	String：末尾の空白を削除する文字列
戻り値	String：末尾の空白を削除した文字列

説明

RTRIM関数は、末尾の空白を削除した文字列を返します。文字列の引数を与えないとエラーになります。

使用例

➲ **使用例**　文字列**"Good morning!"**を返します：

```
RTRIM('Good morning!  ')
```

注：　使用例の関数引数には、感嘆符 (!) の後に2個の空白があります。

関連関数

LTRIM

RIGHT

SUBSTRING

TRIM

4.100 SECOND

目的

時刻の中の秒を返します。

構文

SECOND （ *時刻* ）

時刻	Time：秒を求める時刻
戻り値	Integer：*時刻* の中の秒

説明

時刻の中の秒を整数値0～59で返します。

使用例

➲ **使用例**　秒の整数**12**を返します：

```
SECOND('10:11:12')
```

関連関数

HOUR

MINUTE

HMS

4.101 SECS_BETWEEN

目的

二つの時刻の間の秒数を返します。

構文

時刻1	Time：時刻間の秒数を計算する第一の時刻
時刻2	Time：時刻間の秒数を計算する第二の時刻
戻り値	Integer：*時刻1*と*時刻2*の間の秒数

説明

二つの時刻の間の秒数を返します。*時刻1の引数*は*時刻2の引数*の前でも後でもかまいません。

使用例

➲ **使用例**　秒数**36000**を返します：

```
SECS_BETWEEN('10:10:10', '20:10:10')
```

関連関数

DAYS_BETWEEN

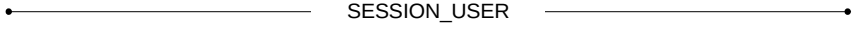
TIMESTAMPDIFF

4.102 SESSION_USER

目的

DBMasterに接続している現在のユーザーを返します。

構文



戻り値　　現在のセッションのユーザー

説明

DBMasterに接続している現在のユーザーを返します。

使用例

➲ **使用例**　現在のセッション・ユーザーを返します：

```
INSERT INTO t1 VALUES (SESSION_USER);
SELECT SESSION_USER;
SELECT c1 FROM t1 WHERE c2 = SESSION_USER;
```

関連関数

CURRENT_USER

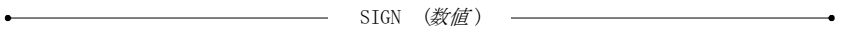
USER

4.103 SIGN

目的

数値の符号(＋/－)を判定します。

構文



数値	Double：符号を判定する数値
戻り値	Integer：*数値*の符号を表す値

説明

数値の符号を表す整数値を返します。正符号ならば+1、ゼロならば0、負符号ならば-1が返されます。

使用例

- **使用例1**　正符号の**1**を返します：

```
SIGN(12.3)
```

- **使用例2**　ゼロの**0**を返します：

```
SIGN(0)
```

- **使用例3**　負符号の**-1**を返します：

```
SIGN(-12.3)
```

関連関数

ABS

4.104 SIN

目的

サインの値を返します。

構文

SIN（*数値*）

数値	Double：サインを求める数値
戻り値	Double：*数値* のサイン

説明

ラジアンで表した数値のサインを倍精度浮動小数点数で返します。

使用例

➲ **使用例　4.79425538604203e-001** を返します：

```
SIN(0.5)
```

関連関数

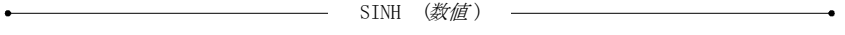
ACOS	COS	SINH
ASIN	COSH	TAN
ATAN	COT	TANH
ATAN2	DEGREES	RADIANS

4.105 SINH

目的

ハイパーボリック・サインの値を返します。

構文



数値	Double：ハイパーボリック・サインを求める数値
戻り値	Double：*数値* のハイパーボリック・サイン

説明

ラジアンで表した数値のハイパーボリック・サインを倍精度浮動小数点数で返します。

使用例

➲ 使用例　**5.21095305493747e-001** を返します：

```
SINH(0.5)
```

関連関数

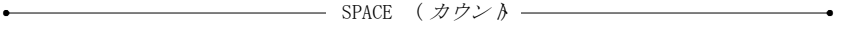
ACOS	COS	SIN
ASIN	COSH	TAN
ATAN	COT	TANH
ATAN2	DEGREES	RADIANS

4.106 SPACE

目的

指定した個数のスペースをもつ空白文字列を作成します。

構文



カウント	Integer：スペースの個数
戻り値	String：*カウント*個のスペースをもつ文字列

説明

SPACE関数は、指定した個数のスペースからなる文字列を返します。カウントが0以下のときはNULLが返されます。

使用例

➲ **使用例1　3**個のスペースの文字列**“　”**を返します：

```
SPACE(3)
```

➲ **使用例2**　文字列**“　Good morning!”**を返します：

```
CONCAT(SPACE(3), 'Good morning!')
```

注：　　戻り値の前には3個のスペースがあります。

関連関数

CONCAT

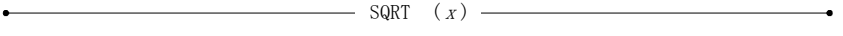
REPEAT

4.107 SQRT

目的

数値の平方根を計算します。

構文



x	Double：平方根を計算する数値
戻り値	Double：*x*の平方根

説明

x の平方根を倍精度浮動小数点数で返します。

使用例

➲ **使用例**　平方根**1.30000000000000e+001**を返します：

```
SQRT(169)
```

4.108 STRTOINT

目的

文字列を整数に変換します。

構文

文字列	String：数値に変換する文字列
戻り値	Bigint：*文字列*を変換した整数の値

説明

STRTOINT関数は文字列を整数に変換します。文字列の引数がNULLのときは、NULLが返されます。文字列の引数が整数に変換できないときは、エラーが返されます。

使用例

➲ **使用例**　整数**1234**を返します：

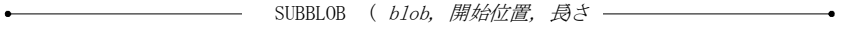
```
STRTOINT('1234')
```

4.109 SUBBLOB

目的

BLOBのデータを取り出し、一時的なBLOB(LONG VARBINARY)としてパッケージします。

構文



blob	BLOB(LONG VARBINARY、LONG VARCHAR、FILE)：データを取り出すオブジェクト
開始位置	Integer：*blob*データの取り出し開始位置
長さ	Integer：取り出すバイト数
戻り値	Long Varbinary：*blob*から取り出した一時的なLONG VARBINARYオブジェクト

説明

*blob*の*開始位置*から*長さ* バイトを取り出して導出した一時的なBLOBを返します。BLOBのバイト位置は1からカウントします。引数がNULLまたは0のときは、以下の規則に従います：

- blob が NULL のときは、NULL が返されます。
- 開始位置または 長さが NULL のときは、blob 全体が一時的な BLOB として返されます。
- 開始位置≤ 0 または 長さ < 0 のときは、NULL が返されます。
- 開始位置 > blob の長さのときは、NULL が返されます。
- 長さが 0 のときは、空列の一時的な BLOB が返されます。

使用例

➲ **使用例　Data BLOBの1001バイトから1100バイトまでを取り出した一時BLOBを返します：**

```
SUBBLOB(Data, 1001, 100)
```

関連関数

SUBBLOBTOBIN

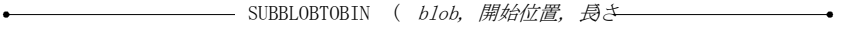
SUBBLOBTOCHAR

4.110 SUBBLOBTOBIN

目的

BLOBのデータを取り出しBINARY(3992) データ型として返します。

構文



blob	BLOB (LONG VARBINARY、LONG VARCHAR、FILE)：データを取り出すBLOB
開始位置	Integer：*blob*のデータを取り出す開始位置
長さ	Integer：取り出すデータのバイト数
戻り値	Binary：*blob*から取り出されたデータ

説明

blobの開始位置 から*長さ* バイト取り出して導出されたBinaryデータを返します。BLOBのバイト位置は1からカウントします。引数がNULLまたは0のときは、以下の規則に従います：

- blobがNULLのときは、NULLが返されます。
- 開始位置または長さが NULLのときは、blob 全体が Binaryデータとして返されます。
- 開始位置≤ 0 または 長さ < 0 のときは、NULL が返されます。
- 開始位置 > blob の長さのときは、NULL が返されます。
- 長さが 0 のときは、空列が返されます。

使用例

➲ **使用例　Data BLOB** の**1001**バイトから**1100**バイトまでのデータを取り出し**BINARY**として返します：

```
SUBBLOBTOBIN(Data, 1001, 100)
```

関連関数

SUBBLOB

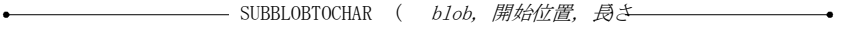
SUBBLOBTOCHAR

4.111 SUBBLOBTOCHAR

目的

BLOBのデータを取り出しVARCHAR(3992)として返します。

構文



blob	BLOB (LONG VARBINARY、LONG VARCHAR、FILE)：データを取り出すBLOB
開始位置	Integer：*blob*のデータを取り出す開始位置
長さ	Integer：取り出すデータのバイト数
戻り値	String：*blob*から取り出したデータの文字列

説明

*blob*の*開始位置* から*長さ* バイト取り出して導出された文字列データを返します。BLOBのバイト位置は1からカウントします。引数がNULLまたは0のときは、以下の規則に従います：

- blob が NULL のときは、NULL が返されます。
- 開始位置または 長さが NULL のときは、blob 全体が文字列データとして返されます。
- 開始位置≤ 0 または 長さ < 0 のときは、NULL が返されます。
- 開始位置 > blob の長さのときは、NULL が返されます。
- 長さが 0 のときは、空列が返されます。

使用例

➲ **使用例　Data BLOB**の**1001**バイトから**1100**バイトまでのデータを取り出し文字列として返します：

```
SUBBLOBTOCHAR(Data, 1001, 100)
```

関連関数

SUBBLOB

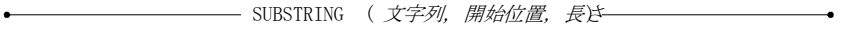
SUBBLOBTOBIN

4.112 SUBSTRING

目的

文字列の中から部分文字列を取り出します。

構文



文字列	String：部分文字列を取り出す文字列
開始位置	Integer：部分文字列の取り出し開始位置
長さ	Integer：取り出す文字数
戻り値	String：*文字列*から取り出された部分文字列

説明

SUBSTRING関数は、*文字列* の *開始位置* から *長さ* だけ取り出した部分文字列を返します。文字列の位置は1からカウントします。引数がNULLまたは0のときは、以下の規則に従います：

- 文字列が NULL のときは、NULL が返されます。
- 開始位置または 長さが NULL のときは、文字列と同じデータが返されます。
- 開始位置＜0 または 長さ＜0 のときは、NULL が返されます。

- 開始位置≥ 文字列の長さのときは、NULL が返されます。
- 長さが 0 のときは、空列が返されます。

使用例

➲ **使用例**　文字列**“morning”**を返します：

```
SUBSTRING('Good morning!', 6, 7)
```

関連関数

LEFT

LOCATE

LTRIM

RIGHT

RTRIM

TRIM

4.113 TAN

目的

タンジェントの値を返します。

構文

TAN （*数値*）

数値	Double：タンジェントの値を求める数値
戻り値	Double：*数値* のタンジェント

説明

ラジアンで表された数値のタンジェントの値を倍精度浮動小数点数で返します。

使用例

➲ **使用例　5.46302489843790e-001**を返します：

```
TAN(0.5)
```

関連関数

ACOS	COS	SIN
ASIN	COSH	SINH
ATAN	COT	TANH
ATAN2	DEGREES	RADIANS

4.114 TANH

目的

ハイパーボリック・タンジェントの値を返します。

構文

TANH （*数値*）

数値　　Double：ハイパーボリックタンジェントの値を求める数値

戻り値　　Double：*数値*のハイパーボリック・タンジェント

説明

ラジアンで表された数値のハイパーボリック・タンジェントの値を倍精度浮動小数点数で返します。

使用例

➲ **使用例　4.62117157260010e-001**を返します：

```
TANH(0.5)
```

関連関数

ACOS	COS	SIN
ASIN	COSH	SINH
ATAN	COT	TAN
ATAN2	DEGREES	RADIANS

4.115 TIMEPART

目的

タイムスタンプの中の時刻を返します。

構文

TIMEPART （ *タイムスタンプ*

タイムスタンプ	Timestamp：時刻を取り出すタイムスタンプ
戻り値	Time：*タイムスタンプ*の中の時刻

説明

タイムスタンプの中の時刻を返します。

使用例

➲ **使用例**　時刻**10:11:12**を返します。

```
TIMEPART('1996-02-29 10:11:12.123')
```

関連関数

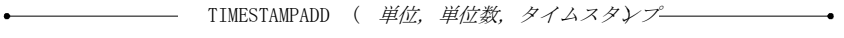
DATEPART

4.116 TIMESTAMPADD

目的

指定した単位の単位数をタイムスタンプの時刻または日付に加えます。

構文



単位	String：加える単位
数値	Integer：加える数値
タイムスタンプ	Timestamp：*数値* を加えるタイムスタンプ
戻り値	Timestamp：*数値* を加えたタイムスタンプ

説明

指定した単位の単位数を加えたタイムスタンプを返します。

単位 (ODBC プログラムの指定)	単位数
“f” (SQL_TSI_FRAC_SECOND)	小数点以下の秒数
“s” (SQL_TSI_SECOND)	秒数
“m” (SQL_TSI_MINUTE)	分数
“h” (SQL_TSI_HOUR)	時間数
“D” (SQL_TSI_DAY)	日数
“W” (SQL_TSI_WEEK)	週数
“M” (SQL_TSI_MONTH)	月数
“Q” (SQL_TSI_QUARTER)	四半期数
“Y” (SQL_TSI_YEAR)	年数

使用例

➲ **使用例**　**20** 時間を加えたタイムスタンプ **1996-01-17 06:10:10** を返します：

```
TIMESTAMPADD('h',20,'1996-01-16 10:10:10')
```

関連関数

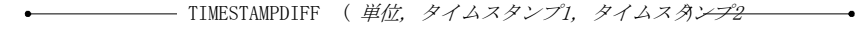
TIMESTAMPDIFF

4.117 TIMESTAMPDIFF

目的

二つのタイムスタンプの差を指定した単位で返します。

構文



単位	String：タイムスタンプの差の単位
タイムスタンプ1	Timestamp：差を求める第一タイムスタンプ
タイムスタンプ2	Timestamp：差を求める第二タイムスタンプ
戻り値	Double：*タイムスタンプ2 - タイムスタンプ1* の値

説明

*タイムスタンプ2 - タイムスタンプ1*の差の単位数を返します。戻り値が負になることもあります。

単位 (ODBC プログラムの指定)	単位数
“f” (SQL_TSI_FRAC_SECOND)	小数点以下の秒数
“s” (SQL_TSI_SECOND)	秒数
“m” (SQL_TSI_MINUTE)	分数
“h” (SQL_TSI_HOUR)	時間数
“D” (SQL_TSI_DAY)	日数
“W” (SQL_TSI_WEEK)	週数
“M” (SQL_TSI_MONTH)	月数
“Q” (SQL_TSI_QUARTER)	四半期数
“Y” (SQL_TSI_YEAR)	年数

使用例

➲ **使用例**　タイムスタンプの差を**2.40000000000000e+001**時間で返します：

```
TIMESTAMPDIFF('h','1996-01-16 10:10:10', '1996-01-17 10:10:10')
```

関連関数

TIMESTAMPADD

DAYS_BETWEEN

SECS_BETWEEN

4.118 TO_DATE

目的

TO_DATE機能は選択されたストリングをDATEフォーマットに変換します。ストリングは任意のデータ・タイプでかまいませんが、日付に変換された時有効な日付にならなければいけません。TO_DATE機能は2つのパラメーター、string_expr および date_format_string から成ります。date_format_stringがDATEデータ型の結果セットがとるフォーマットを表わしている一方、string_exprパラメーターは変換されるストリングを表わします。

TO_DATE UDF はDBMasterのインストールディレクトリ\shared\udfにあります。データベースに自動に作成できません。使用する時、手動で作成できます。

以下のコマンドを実行する:

```
create function to_date.TO_DATE(varchar(20), varchar(20)) RETURNS DATE;
```

構文

TO_DATE (*string_expr*, *date_format_string*)

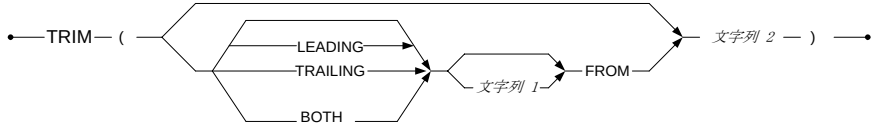
string_expr	変換するストリング表現
date_format_string	日付フォーマットがとるべきフォーマット。**Y**または**y**を使用して年を、**M**または**m**を使用して月を、**D**あるいは**d**を使用して日を表示してください。**/**あるいは**–**をセパレーターとして使用してください。
戻り値	DATEデータ型ストリングとして返されたストリング表現。

4.119 TRIM

目的

文字列の先頭と末尾の文字を削除します。

構文



LEADING	*trim_sourceの前の*trim_stringを削除する
TRAILING	*rim_sourceの後の*trim_stringを削除する
BOTH	*rim_sourceの前後の*trim_stringを削除する

何も選択しないと(leading, trailing, both)、トリム関数は*rim_sourceの前後の*trim_stringを削除します。

文字列1　　*文字列2から削除するキャラクタです。*

この引数はコミットされると。トリム関数は*trim_source.*から全てのLEADING、TRAILINGスペースを削除します。

文字列2　　トリムする文字列。

戻り値　　String：*文字列2*から*文字列1*をトリムした文字列

説明

TRIM関数は、LTRIM関数とRTRIM関数をあわせたものです。複数の文字を*文字列1*に指定することができます。各文字は有効なトリム文字として見なされます。

LEADING、TRAILING、BOTHオプションのいずれも指定しない場合、初期設定のトリムオプションはBOTHです。初期設定の*文字列1*は、空白(' ')です。*文字列1*が空の文字列('')の場合、戻り値は*文字列2*のままになります。*文字列2*がNULLの場合、どのオプションと*文字列1*を使用したかに関わらず、結果もNULLになります。以下の例で示したように、LENGTH関数をTRIM関数と共に使用することができます。

使用例

➲ **使用例1**

```
SELECT TRIM(both 'a' FROM 'aabcaa');
TRIM(BOTH 'A' FROM 'AABCAA')
============================
bc
```

➲ **使用例2**

```
SELECT TRIM(FROM 'aabcaa');
TRIM(FROM 'AABCAA')
===================
aabcaa
```

➲ **使用例3**

```
SELECT TRIM('a' FROM 'aabcaa');
TRIM('A' FROM 'AABCAA')
=======================
```

```
bc
```

➲ 使用例4

```
SELECT TRIM('abc' FROM 'abckjkjjdcba');
TRIM('ABC' FROM 'ABCKJKJJDCBA')
================================
kjkjjd
```

➲ 使用例5

```
SELECT TRIM ('a c' FROM 'ac ddbc');
TRIM ('A C' FROM 'AC DDBC')
===========================
ddb
```

➲ 使用例6

```
SELECT LENGTH(TRIM(LEADING FROM ' abc '));
LENGTH(TRIM(LEADING FROM ' ABC '))
==============================
               3
```

➲ 使用例7

```
SELECT LENGTH(TRIM(LEADING 'a' FROM 'aabc '));
LENGTH(TRIM(LEADING 'A' FROM 'AA'))
==============================
               2
```

➲ 使用例8

```
SELECT LENGTH(TRIM (TRAILING FROM 'aabc '));
LENGTH(TRIM(TRAILING FROM 'AABC'))
==============================
              4
```

➲ 使用例9

```
SELECT LENGTH(TRIM (TRAILING 'a' FROM 'aabcaa'));
```

```
LENGTH(TRIM(TRAILING 'A' FROM 'AABCAA'))
==============================
             4
```

関連関数

LTRIM

RTRIM

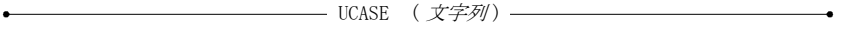
SUBSTRING

4.120 UCASE

目的

大文字の文字列に変換します。

構文



文字列	String：大文字に変換するテキスト
戻り値	String：*文字列*を大文字に変換したテキスト

説明

UCASE関数は、*文字列*の引数の全ての小文字を大文字に変換します。引数がNULLのときはNULLが返されます。引数を指定しないとエラーになります。

使用例

➲ **使用例1**　文字列**“ABCDEF”**を返します：

```
UCASE('ABCdef')
```

➲ **使用例2**　文字列**“ABC123”**を返します：

```
UCASE('abc123')
```

➲ **使用例3**　文字列**“ABC@#$”**を返します：

```
UCASE('abc@#$')
```

関連関数

LCASE

LOWER

UPPER

4.121 UPPER

目的

大文字の文字列に変換します。

構文

UPPER（*文字列*）

文字列	String：大文字に変換するテキスト
戻り値	String：*文字列*を大文字に変換したテキスト

説明

UCASEと同様の働きをします。大文字の文字列に変換します。NULL文字列の引数はNULLを返します。

使用例

➲ **使用例1**

```
SELECT UPPER('abcdef');

UPPER('ABCDEF')

================

ABCDEF
```

関連関数

LCASE

LOWER

UCASE

4.122 USER

目的

現ユーザーの名前を返します。

構文

戻り値　　String：現ユーザーの名前

説明

現ユーザーの認可名を返します。認可名は、SQL_USER_NAME オプションを付けてSQLGetInfoを呼び出しても取得することができます。

関連関数

CURRENT_USER

DATABASE

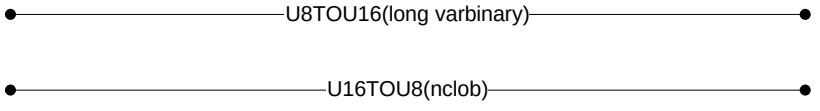
SESSION_USER

4.123 UTFConvert

目的

UTFConvert関数はUTF-8とUTF-16文字を交互に変換します。U8TOU16とU16TOU8の二つの関数が含まれます。

構文



long varbinary　UTF-16に変換されるUTF-8テキストデータ

nclob　UTF-8に変換されるUTF-16テキストデータ

説明

UTFConvert関数はDBMasterのインストールディレクトリ\shared\udfに存在します。DBMasterの初期状態で使用できるものではなく、必要に応じてユーザが作成する必要があります。作成には使用例を参照のこと。

使用例

➲ **使用例**　関数作成用コマンド実行例：

```
create function UTFConvert.U8TOU16(long varbinary) returns nclob;
create function UTFConvert.U16TOU8(nclob) returns long varbinary;
```

4.124 WEEK

目的

日付の年初からの週数を返します。

構文

WEEK （*日付*）

日付	Date：年初からの週数を求める日付
戻り値	Integer：*日付*の年初からの週数

説明

日付の年初からの週数を**1**～**53**の整数で返します。

使用例

➲ **使用例**　**1996-02-11**は**1996**の第**5**週なので整数**5**を返します：

```
WEEK('1996-02-01')
```

関連関数

DAYOFMONTH

MONTH

QUARTER

YEAR

4.125 XLSTOTXT

目的

Excelドキュメントを一時的なBLOB に変換します。

構文

XLSTOTXT (*blob*)

Blob:　テキストに変換するカラム名

戻り値: テキストに変換可能な場合、一時BLOBはNCLOBで表示されます

説明

XLSTOTXT 関数関数はBLOBをUnicodeのBlobテキストを含む一時BLOBに変換します。一時 blob またはNULL を返します。DBMaster5.3のUDFはoffice 2007- 2010 バージョンでサポートします。

使用例

➲ **使用例　“memo”**カラムをテキストに変換します

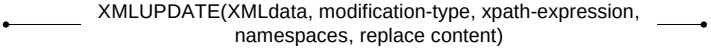
```
XLSTOTXT(memo)
```

4.126 XMLUPDATE

目的

更新xml データを指定します

構文



Xmldata……更新されるXMLデータ

Xpath-expression……更新するXMLデータの位置を指定

Namespaces……オプションでXPath-expressionに使用されるnamespaceを指定します

Replace-content……XPathで指定された内容を変更する値

Return value……更新後のXMLドキュメント全体

説明

xmlupdate関数はXPathを使用しXML更新部分を指定します。

4.127 YEAR

目的

日付の中の年を返します。

構文

YEAR （*日付*）

日付	Date：年を求める日付
戻り値	Integer：*日付*の中の年

説明

日付の中の年を整数**1**～**9999**で返します。

使用例

➲ **使用例**　年の整数**2002**を返します：

```
YEAR('2002-02-01')
```

関連関数

DAYOFMONTH

WEEK

MONTH

QUARTER

5 システム・ストアド・プロシージャ

*システム・ストアド・プロシージャ*は、呼び出してロードすることができるダイナミック・ライブラリ・モジュールです。システム・ストアド・プロシージャには、共有オブジェクトやXMLインポートやXMLエクスポート・プロシージャがあります。

共有オブジェクトは、データベースの共有メモリ(DCCA)に存在する符号付きの可変整数です。共有オブジェクトへのアクセスは、より効率的で、トランザクションから独立しています。データ・レコードと異なり、共有オブジェクトは、データベース・ファイルには保存されません。そのため、共有オブジェクトは、それが削除された時、或いはデータベースを終了すると無くなります。データベースに接続しているどのユーザーも、（SYSADMがそれを追加した後）共有オブジェクトを見ることができます、他のユーザーがロックしない限り、それぞれの共有オブジェクトの値をセットしたり、取得したりすることができます。共有オブジェクトは、符号付きの整数です（4バイト）。どのユーザーも、共有オブジェクトに対し、同等の権限/許可を持っています。つまり、ロック権限を除き、オブジェクトの設定を上書き、或いはリセットできます。

他の2つのシステム・ストアド・プロシージャ（XMLEXPORTとXMLIMORT）は、SYSADMとDBAによってのみ使用され、XMLファイルのインポートとエクスポートを行います。

5.1 COPYTABLE

目的

COPYTABLEのシステムストアドプロシージャはあるテーブルの定義とデータを他のテーブルにコピーするために使用されます。COPYTABLEはソーステーブルのインデックス、テーブル/カラム制約、トリガ、データが出力先のテーブルにコピーされます。

COPYTABLEのシステムストアドプロシージャは自動コミットがオンの時に実行されなければなりません。出力先のテーブルがすでに存在している場合はエラーが返されます。ユーザーがインデックスフラグの名前の変更に1を設定すると、テーブル名が前置されている場合、インデックス名も新しいテーブル名に変更されます。ユーザーがコミットカウントを設定している場合、コマンドはn番目のデータを出力先のテーブルにコピーするたびにコミットするようにコマンドが発行されます。

エラーが発生すると、エラー発生前のすべての処理がコミットされます。COPYTABLEのために実行されなかったコマンドは_spusr.log にレコードされます。ユーザーは dmconfig.iniにDB_SPLOG ディレクトリを指定するか、もしくはアプリケーションの実行ディレクトリ内にあるログをファイルを参照できます。

原型

```
COPYTABLE(VARCHAR(32) source_schema_name INPUT,
VARCHAR(32) source_table_name INPUT,
          VARCHAR(32) destination_schema_name INPUT,
          VARCHAR(32) destination_table_name INPUT,
          VARCHAR(128) tablespace_lock_mode_option_string INPUT,
          VARCHAR(2048) where_condition_string INPUT,
          INT fg_rename_index INPUT,
          INT commit_count INPUT)
```

schema_name : テーブルのスキーマ名、または空白文字列として指定されたときのデフォルトであるカレントユーザーを表します。

table_name: ソースまたは出力先のテーブル名

tablespace_lock_mode_option_string: 文字列中の作成テーブルと同じようにテーブルスペースまたはロックモード文字列で指定されます。構文に指定される識別子はSQL構文規則に従わなければなりません。

where_condition_string: 構文のSELECT構文と同じようにwhere条件で指定されます。 構文内で指定される識別子はSQL構文規則に従ったものでなければなりません。

fg_rename_index: ソースのインデックス名の前にテーブル名がある場合インデックスの名前の変更するかどうかを指定するフラグです。有効値は0または1です。

commit_count: nレコードが挿入されるたびにコミットします。有効値は0からnまでです。

使用例

➲ **例**

以下の文字列はScoresテーブルを「Math score > 70」のテーブルスペースの異なるテーブルScores70にコピーするのに使用されます。インデックスの名前を変更せず、10のレコードごとにコミットします :

```
dmSQL> call copytable('SYSADM', 'Scores', 'SYSADM', 'Scores70', 'in
tablespace1', 'Math > 70', 0, 10);
```

5.2 GETCPUNUMBER

目的

GETCPUNUMBERシステムストアドプロシージャはロジックプロセッサの数を取ります。

原型

```
GETCPUNUMBER (INT CPU_NUMBER OUTPUT)
```

CPU_NUMBER ……出力バラメータ、ロジックプロセッサの数。

説明

GETCPUNUMBERとSETAFFINITYシステムストアドプロシージャを使用して、ユーザーは今のシステム声明を取ることができて、ランタイムにDBMasterを再起動する必要が無く、 接続のCPU アフィニティを設定できます。

使用例

➲ **使用例**　GETCPUNUMBERというCPUの数を取ります：

```
dmSQL> call GETCPUNUMBER(?);
```

5.3 SETAFFINITY

目的

GETCPUNUMBERシステムストアドプロシージャはロジックプロセッサの数を取ります。

原型

```
SETAFFINITY (INT CONNECTION_ID INPUT,  CHAR(64) AFFINITY_MASK INPUT)
```

*CONNECTION_ID　　入力バラメータ、接続のID或いはサーバーです。ユーザーはを*で*"select connection_id from sysuser"*またはシステムモニターをチェックすることを通じでこれを取ります。*windows* にこれはスレッドのID で、*Unix-like* システムにプロセスID です。

AFFINITY_MASK　　入力バラメータ、CPU アフィニティマスク。有効なアフィニティマスクは1或いは0から構成されます。'1' はCPUが有効だという意味で、 '0'は無効だという意味です。

説明

GETCPUNUMBERとSETAFFINITYシステムストアドプロシージャを使用して、ユーザーは今のシステム声明を取ることができて、ランタイムにDBMasterを再起動する必要が無く、接続のCPUアフィニティを設定できます。

CPU アフィニティはアフィニティマスクから定義します。アフィニティマスクはビットベクタで、各ビットはプロセッサを代表します。DBMasterはアフィニティマスクをchar(64)と定義しますので、これは64 CPU を設定します。

使用例

➲ 使用例1

8-CPUシステムにアフィニティマスクの値は以下の通りです。(高い位置に連続する0は無視します)：

Decimal value	Binary bit mask	Allow run on CPU
1	‘1’	0
3	‘11’	0 and 1
7	‘111’	0, 1, and 2
15	‘1111’	0, 1, 2, and 3
31	‘11111’	0, 1, 2, 3, and 4
63	‘111111’	0, 1, 2, 3, 4, and 5
127	‘1111111’	0, 1, 2, 3, 4, 5, and 6
255	‘11111111’	0, 1, 2, 3, 4, 5, 6, and 7

➲ 使用例2

ユーザーはアフィニティマスクを設定する前、システム情報をとる必要があります、例えば、サーバーのCPUの数、接続のCPU用法、正確的なアフィニティマスク。

GETCPUNUMBERを呼び出してCPUの数を取ります：

```
dmSQL> call GETCPUNUMBER(?);
```

接続のCPU用法、正確的なアフィニティマスクを取ります：

```
dmSQL> select connection_id, affinity_mask, priority_level, cpu_usage from
sysuser;
```

CPUアフィニティを設定します。CPU 0 と1にこの接続が実行することが許せます：

```
dmSQL> select connection_id , user_name from sysuser;


    CONNECT*              USER_NAME
================   ======================

30420                  BACKUP_SERVER

30418                  SYSADM


2 rows selected

DMSQL> call setaffinity(30418,'11');
```

精確な接続にクエリsysuserを通じてCPU アフィニティを取ります：

```
dmSQL> select affinity_mask from sysuser where connection_id = ?;
```

5.4 SETPRIORITY

目的

スレッドとプロセッサーの優先級を設定します。

原型

```
GETCPUNUMBER (INT CONNECTION_ID INPUT,INT PRIORITY_LEVEL INPUT)
```

CONNECTION_ID…入力引数、接続のID或いはサーバーです。ユーザーはをで"select connection_id from sysuser"またはシステムモニターをチェックすることを通じでこれを取ります。windows にこれはスレッドのIDで、Unix-like システムにプロセスIDです。

PRIORITY_LEVEL……入力引数、五つのレベルがあります。ディフォルト優先レベルは3です。有効な優先級は'1'、 '2'、 '3'、 '4'と'5'で、'1'は最低

レベル、‘2’は低いレベル、‘3’は普通レベル、‘4’は高いレベル、‘5’は最高優先級という意味です。

説明

SETPRIORITYシステムストアドプロシージャを使用して、ユーザーはランタイムにDBMasterを再起動する必要が無く、接続の優先を設定できます。Linux にroot権限が必要ですので、ユーザーは高い優先級を設定することができません。だから、Linux に低い優先が設定できます。でも、Windows に制限がありません。

使用例

➲ **使用例**

ユーザーはアフィニティマスクを設定する前、システム情報をとる必要があります、例えば、サーバーのCPUの数、接続のCPU用法、優先級。

GETCPUNUMBERを呼び出してCPUの数を取ります：

```
dmSQL> call GETCPUNUMBER(?);
```

接続のCPU用法、優先級を取ります：

```
dmSQL> select connection_id, affinity_mask, priority_level, cpu_usage from
sysuser;
```

優先級を設定します：

```
dmSQL> select connection_id , user_name from sysuser;


    CONNECT*              USER_NAME
===============   ======================
30420                  BACKUP_SERVER
30418                  SYSADM


2 rows selected
DMSQL> call setpriority(30418,3);
```

精確な接続にクエリsysuserを通じて優先級を取ります：

```
dmSQL> select priority_level from sysuser where connection_id = ?;
```

5.5 SETSYSTEMOPTION

目的

ランタイムにシステムオプションを設定します。つまり、データベースが実行している中にSetSystemOptionストアドプロシージャを使用してBKSVR、BKDIR、BKITV、DBKTV、BKTIM、BKFUL、BKFOM、BKZIP、BKCMP、BKRTS、BKCHK、FBKTM、FBKTV、DBKMX、BKODR、BKFRM、STSSPを変更されることができます。

原型

```
setSystemOption('optionName', 'optionValue');
```

optionName　　*システムオプションの名*

optionValue　　*システムオプションの値*

使用例

➲ **使用例1**

バックアップサーバーを起動します：

```
dmSQL> CALL SETSYSTEMOPTION('BKSVR','1');
```

➲ **使用例2**

バックアップサーバーを起動すると、**dmconfig.ini** ファイルの適当バックアップ引数を設定します。

```
dmSQL> Call SetSystemOption('STARTBACKUP','1'); //完全バックアップを実行します
dmSQL> Call SetSystemOption('STARTBACKUP','2'); //増分バックアップを実行します
dmSQL> Call SetSystemOption('STARTBACKUP','3'); //差分バックアップを実行します
```

➲ **使用例3**

ランタイムに更新統計サンプルを60に設定します：

```
dmSQL> call setSystemOption('STSSP', '60');
```

➲ **使用例4**

実行中の更新統計を終了します：

```
dmSQL> call setSystemOption('STS_ABORT', '14076'); //実行している接続IDは14076である更新統計を終了します。

dmSQL> call setSystemOption('STS_ABORT', '0'); //値0は特別な接続IDです。更新統計と関連する全部のコマンドと終了するという意味です。
```

5.6 SETSYSTEMOPTIONW

目的

ランタイムにシステムオプションを設定します、そして設定をdmconfig.iniファイルに書き入れます。

原型

```
setSystemOptionW('optionName', 'optionValue');
```

optionName　　*システムオプションの名*

optionValue　　*システムオプションの値*

説明

dmconfig引数はsetSystemOptionを呼び出すことを通じてランタイム設定を許すと、setSystemOptionWを通じてランタイムの設定ができます。.

ユーザーはgetSystemOptionを呼び出すことによって新しいオプション値を取ることができます。

使用例

➲ **使用例1**

ユーザーは統計デッモンを更新する場合、optionName をSTSVR、optionValueは1を設定して、ランタイムセットは, "DB_STSVR = 1" にすると、dmconfig.iniファイルまで以下のように書き入れます：

ストアドプロシージャSystemOptionWを呼び出す前、dmconfig.iniファイルは：

```
[DBSAMPLE5]
; Here omit other keywords
DB_STSVR = 0
```

setSystemOptionWを呼び出すことを実行します：

```
dmSQL> call setSystemOptionW('STSVR', '1');
dmSQL> call getSystemOption('STSVR',?);
OPTION_VALUE                    : 1
```

ストアドプロシージャSystemOptionWを呼び出した後、dmconfig.iniファイルは：

```
[DSAMPLE5]
; Here omit other keywords
DB_STSVR = 1
```

5.7 SOADD

目的

共有オブジェクトの値を増やします。

原型

```
SOADD(
      INTEGER SHID,
      INTEGER ADDEND,
      INTEGER NEW_VAL OUTPUT)
```

setid　　共有オブジェクトのID

addend　　追加する正/負の値

new_val　　追加後の値

使用例

➲ **使用例**　共有オブジェクト2に3を追加し、新しい値3を取得します：

```
dmSQL> call SYSADM.SOAdd(2,3,?);
new_val: 3
```

関連プロシージャ

SOCREATE

SODROP

SOLOCK

SOREAD

SOSET

SOUNLOCK

5.8 SOCREATE

目的

共有オブジェクトを作成します。

原型

```
SOCREATE(
        INTEGER SETID,
        INTEGER INIT_VAL,
        INTEGER SHID OUTPUT)
```

setid　　共有オブジェクトの割り当てID

0　　割り当てシステム

otherwise　　割り当てユーザー

init_val 初期値

shid 生成した共有オブジェクトのID

説明

共有オブジェクトを使用するために、SOCreate()を使用して、特殊な識別子と初期値で共有オブジェクトを作成します。更に、識別子を表示することで、SORead()、SOSet()、SOAdd()(共有オブジェクト値をそれぞれ、読み込み、修正、増加します)を使用します。共有オブジェクトがあらゆる接続によってアクセスされるので、同時実行性の制御のためにSOLock()とSOUnlockを使用することができます。共有オブジェクトが使用されなくなった時、SODrop()で削除することができます。

使用例

➲ **使用例1** 初期値0で共有オブジェクト0を作成します：

```
dmSQL> call SYSADM.SOCreate(0,0,?);
Shid: 1
```

➲ **使用例2** 初期値0で共有オブジェクト2を作成します：

```
dmSQL> call SYSADM.SOCreate(2,0,?);
Shid: 2
```

関連プロシージャ

SOADD

SODROP

SOLOCK

SOREAD

SOSET

SOUNLOCK

5.9 SODROP

目的

使われていない共有オブジェクトを削除します。

原型

```
SODROP(INTEGER SHID)
```

shid　　削除する共有オブジェクトのID

使用例

➲ **使用例**　共有オブジェクト1を削除します：

```
dmSQL> call SYSADM.SODrop(1);
```

関連プロシージャ

SOADD

SOCREATE

SOLOCK

SOREAD

SOSET

SOUNLOCK

5.10 SOLOCK

目的

共有オブジェクトをロックします。

原型

```
SOLOCK(INTEGER SHID)
```

shid　　ロックしようとする共有オブジェクトのID

説明

共有オブジェクトがロックされると、他のユーザーは、読み込み、設定、追加、削除、ロック、ロック解除することができません。ロックをかけたユーザーのみが、その他の6つのシステム・ストアド・プロシージャを使用することができます。

使用例

➲ **使用例**　共有オブジェクト1をロックします：

```
dmSQL> call SYSADM.SOLock(1);
```

関連プロシージャ

SOADD

SOCREATE

SODROP

SOREAD

SOSET

SOUNLOCK

5.11 SOREAD

目的

共有オブジェクトの値を読み込み（取得）します。

原型

```
SOREAD(
       INTEGER SHID,
       INTEGER VAL OUTPUT)
```

setid 共有オブジェクトのID

val 共有オブジェクトの値

使用例

➲ **使用例** 共有オブジェクト2の値を取得します：

```
dmSQL> call SYSADM.SORead(2,?);
val: 3
```

関連プロシージャ

SOADD

SOCREATE

SODROP

SOLOCK

SOSET

SOUNLOCK

5.12 SOSET

目的

共有オブジェクトの値を修正します。

原型

```
SOSET(
```

```
    INTEGER SHID,
    INTEGER NEW_VAL,
    INTEGER OLD_VAL OUTPUT)
```

Setid 共有オブジェクトのID

new_val 割り当てる値

old_val 割り当てる前の値

使用例

➲ **使用例** 共有オブジェクト2の値を-2にセットします：

```
dmSQL> call SYSADM.SOSet(2,-2,?);
old_val: 3
```

関連プロシージャ

SOADD
SOCREATE
SODROP
SOLOCK
SOREAD
SOUNLOCK

5.13 SOUNLOCK

目的

共有オブジェクトのロックを解除します。

原型

```
SOUNLOCK(INTEGER SHID)
```

shid ロック解除しようとする共有オブジェクトのID

説明

共有オブジェクトがロックされると、他のユーザーは、読み込み、設定、追加、削除、ロック、ロック解除をすることができません。共有オブジェクトにロックをかけたユーザーのみが、ロックを解除することができます。

使用例

➲ **使用例** 共有オブジェクト1のロックを解除します：

```
dmSQL> call SYSADM.SOUnlock(1);
```

関連プロシージャ

SOADD
SOCREATE
SODROP
SOLOCK
SOREAD
SOSET

5.14 XMLEXPORT

目的

データをDBMasterからXMLにエクスポートします。

原型

```
XMLEXPORT(
                                VARCHAR(256)                    FILE_PATH,
                                     VARCHAR(256)                 DB_TAG,
                                VARCHAR(256)                    XML_HEADER,
                                VARCHAR(16000)                  OBJECT_STR,
                                 VARCHAR(256)                   OPTION_STR,
        VARCHAR(256)  LOG_PATH)
```

説明

XMLEXPORTシステム・ストアド・プロシージャは、データをDBMasterからXMLにエクスポートすることができるプログラム可能なインターフェースです。SYSADMとDBAのみが、このストアド・プロシージャを呼び出す

ことができます。又、XMLEXPORTはシステム・ストアド・プロシージャですので、他のユーザーに実行権限を与えることはできません。

XMLEXPORTは、DBMasterデータベースからXMLファイルに表をエクスポートし、対応するストアド・プロシージャを1つ呼び出すことで、複数の表を処理することができます。XMLファイルの内容とDBMasterの表の間のマッピングについての解説は、ディスクリプション文字列の中で指示します。このディスクリプション文字列は、ストアド・プロシージャに与えられた引数の1つとして用いられます。

引数名	データ型	サイズ(バイト)	解説	大文字と小文字の区別
file_path	varchar	256	エクスポートしたXMLファイルの絶対パス	オペレーティング・システムに拠ります。
db_tag	varchar	128	カスタマイズされたデータベースのタグ	区別します(出力は、同じ文字タイプになります)。
xml_header	varchar	256	カスタマイズされたXMLのヘッダー	区別します(出力は、同じ文字タイプになります)。
object_str	varchar	16000	エクスポートされたオブジェクト用のディスクリプション文字列	DBMasterの設定に拠ります。
option_flag	varchar	256	オプション・フラグ用のディスクリプション文字列	区別しません。
log_path	varchar	256	クライアント側のエラー・ログファイルの絶対パス	オペレーティング・システムに拠ります。

図5-1 XMLEXPORTの引数

XMLEXPORTに引数を与える

データベースからエクスポートされたXMLファイルは、サーバーで生成する必要があります。最初のfile_pathには、絶対パスの文字列を指定します。使用例5を参照してください。

2つ目のdb_tagは、タグをカスタマイズするために使用します。NULLや空の文字列が指定された場合、初期設定値が使用されます。

3つ目のxml_headerは、XMLファイルのヘッダーをカスタマイズするために使用します。XMLファイルのヘッダーにスタイル情報、ネームスペース定義、コメント等の文字列を追加できます。出力ファイルには必ず<?xml version=”1.0”>がありますので、重複させる必要はありません。

4つ目の引数は、object_strです。

```
Object_str=:
{ <element> [; <element>…]

<element>=:
{TABLE_NAME | <select_query>} [#TABLE_TAG]
```

<element>は表を表します。<element>間に使用するデリミタは、セミコロン(;)です。<element>からの最初のトークンが、「SELECT」（大文字と小文字を区別しない）の場合、この<element>は<select_query> [#TABLE_TAG]と解釈されます。

或いは、この<element>はTABLE_OWNER.TABLE_NAME [#TABLE_TAG]という意味になります。<element> = TABLE_OWNER. TABLE_NAME [#TABLE_TAG]の場合、この表にある全カラムが選択され、どのカラム・タグもカスタマイズされません。つまり、エクスポートされたXMLファイルのカラム・タグの名前は、対応する表のカラム名と同じです。

TABLE_TAGは、カスタマイズした表のタグを指定するためのものです。TABLE_TAGが無い場合、データベースの表名が、表のタグ名として使用されます。

カラムのタグをカスタマイズする場合、<element>文字列に<select_query>[#TABLE_TAG]のみを使うことができます。カスタマイズされたカラムのタグ名は、<select_query>文の中でカラムの別名を用いることで指定します。<select_query>の中では、「select c1 as name, c2 as type from SYSADM.t2」のように、ASを使います。そうすると、エクスポートされたXMLファイルで、カラムc1は「name」タグになり、カラムc2は「type」タグになります。

5つ目に、option_flagを使って、オプション文字列でオプションを定義します。各オプションは、セミコロン(;)で仕切ります。例えば、カラム名を属性として扱いたい場合、オプション文字列に「column_as_attribute」を使用します。特定のオプションを指定しない場合、オプションは設定されません。オプション・フラグの文字列は、大文字と小文字を区別しません。

オプション・フラグ	設定	設定無し
blob_in_separate_file	BLOB/CLOBカラムのデータは、XMLファイルとは別の一時ファイルとしてエクスポートします。この一時ファイルの名前は、エクスポートDTDに記録されます。	BLOB/CLOBカラムのデータは、XMLファイルの一部としてエクスポートされます。
column_as_attribute	カラムは、XMLファイルの要素としてでは無く、属性としてエクスポートされます。	カラムは、XMLファイルの要素としてエクスポートされます。

オプション・フラグ	設定	設定無し
capitalize_tag_name	タグ名は、XMLファイルで全て大文字になります。	タグ名の大文字と小文字は、データベースの対応する名前のルールと同じになります。
file_type_as_link	FILEデータ型の要素は、エクスポートされません。ファイル名の名前のみがXMLファイルにエクスポートされます。	FILEデータ型の要素は、XMLファイルの一部としてエクスポートされます。
no_schema_dtd	XMLファイル生成の際にスキーマDTDを生成しません。	XMLファイル生成の際に対応するDTDを生成します。

図5-2 XMLEXPORTのオプション

最後のlog_pathは、XMLファイルのエクスポートの際に生成されたログファイルが保存されるクライアント側のパス名です。

XMLファイルをエクスポートする

*Customer*という名前のDBMasterのデータベースから、2つの表（*card*と*contact*）を/usr/john/xmlexport.xmlファイルにエクスポートすると想定します。xmlexport.xmlファイルには、データベースのタグとして「EMPLOYEE」を、表tb_card用のタグとして「TITLE」を、表tb_contact用のタグとして「NUMBER」をそれぞれ独自に使用することとします。

更に、表**tb**_cardのカラムID, FNAME, LNAME 、WORKには独自にカラム・タグとして、それぞれにID_NO、FIRST_NAME、LAST_NAME、JOBを用います。表**tb**_contactのカラムには、独自のタグは使いません。また、XMLファイルの全てのタグ名を大文字にし、BLOBカラムのデータがある場合は、全て別の一時ファイルに保存します。最後に、ログファイル名

は、/client/john/xmlexport.logとして保存します。これら2つの表のレコードは、以下のとおりです。

```
dmSQL> SELECT * FROM tb_card;
 ID                FNAME                    LNAME                    WORK
========== ==================== ==================== ====================
 1           Eddie                Chang                Manager
 2           Hook                 Hu                   SoftwareEngineer
 3           Jackie               Yu                   SoftwareEngineer
 8           Jerry                Liu                  Manager

dmSQL> SELECT * FROM tb_contact;
 NO            FIRST_NAME             LAST_NAME              PHONE
========== ==================== ==================== ====================
 1          Eddie                Chang                2145678
 2          Hook                 Hu                   2335678
 3          Jackie               Yu                   2346678
 4          Jerry                Liu                  2345671
```

➲ XMLファイルにエクスポートする

1. File_pathは、エクスポートするXMLファイルの絶対パスです。ファイルはサーバー側で生成されますので、指定したファイルのパスもサーバー側にする必要があります。この引数に、文字列「/usr/john/xmlexport.xml」を指定します。

2. db_tagは、カスタマイズするデータベースのタグです。NULLや空の文字列は、初期設定値が使われることを意味します。文字列「EMPLOYEE」をこの引数に指定します。

3. xml_headerは、xmlファイルのヘッダーにコメント等を追加する場合に指定します。この例では指定しません（空白）。

4. この例では、次のobject_str文字列を使います。

```
'select ID as ID_NO, FNAME as FIRST_NAME, LNAME as LAST_NAME, WORK as
JOB from SYSADM. tb_card#TITLE;SYSADM. tb_contact#NUMBER'
```

5. オプション文字列の引数には、「capitalize_tag_name;blob_in_separate_file」を指定します。

6. ログのパスの引数には、「/client/john/xmlexport.log」を指定します。

7. 次のようにXMLExportを呼び出します。

```
call  XMLExport (

'/usr/john/xmlexport.xml',

'EMPLOYEE',

'',

'select ID as ID_NO, FNAME as FIRST_NAME, LNAME as LAST_NAME, WORK as
JOB from SYSADM. tb_card#TITLE;SYSADM. tb_contact#NUMBER',

'capitalize_tag_name;blob_in_separate_file',

'/client/john/xmlexport.log');
```

8. エクスポートされたxmlexport.xmlファイルの一部。

```
<EMPLOYEE>

  <TITLE>

    <IO_NO>1</ID_NO>

    <FIRST_NAME>Eddie</FIRST_NAME>

    <LAST_NAME>Chang</LAST_NAME>

    <JOB>Manager</JOB>

  </TITLE>

  <TITLE>

    <ID_NO>2</ID_NO>
```

```
  <FIRST_NAME>Hook</FIRST_NAME>
  <LAST_NAME>Hu</LAST_NAME>
  <JOB>SoftwareEngineer</JOB>
</TITLE>
<TITLE>
  <ID_NO>3</ID_NO>
  <FIRST_NAME>Jackie</FIRST_NAME>
  <LAST_NAME>Yu</LAST_NAME>
  <JOB>SoftwareEngineer</JOB>
</TITLE>
<TITLE>
  <ID_NO>4</ID_NO>
  <FIRST_NAME>Jerry</FIRST_NAME>
  <LAST_NAME>Liu</LAST_NAME>
  <JOB>Manager</JOB>
</TITLE>
<NUMBER>
  <ID_NO>1</ID_NO>
  <FIRST_NAME>Eddie</FIRST_NAME>
  <LAST_NAME>Chang</LAST_NAME>
  <PHONE>2145678</PHONE>
</NUMBER>
<NUMBER>
  <NO>2</NO>
```

```
    <FIRST_NAME>Hook</FIRST_NAME>
    <LAST_NAME>Hu</LAST_NAME>
    <PHONE>2335678</PHONE>
  </NUMBER>
  <NUMBER>
    <NO>3</NO>
    <FIRST_NAME>Jackie</FIRST_NAME>
    <LAST_NAME>Yu</LAST_NAME>
    <PHONE>2346678</PHONE>
  </NUMBER>
  <NUMBER>
    <NO>4</NO>
    <FIRST_NAME>Jerry</FIRST_NAME>
    <LAST_NAME>Liu</LAST_NAME>
    <PHONE>2345671</PHONE>
  </NUMBER>
</EMPLOYEE>
```

➲ **或いは次のようにXMLExportを呼び出します。**

1. オプション文字列に「column_as_attribute」を加えて、XMLExportを呼び出します。 この場合、同時に「blob_in_separate_file」オプションは指定できません。

```
call  XMLExport (
'/usr/john/xmlexport.xml',
'EMPLOYEE',
```

```
'',
'select ID as ID_NO, FNAME as FIRST_NAME, LNAME as LAST_NAME, WORK as
JOB from SYSADM. tb_card#TITLE ',
'capitalize_tag_name;column_as_attribute',
'/client/john/xmlexport.log');
```

2. 結果の一部は以下のとおりです。

```
<EMPLOYEE>
  <TITLE ID_NO="1" FIRST_NAME="Eddie" LAST_NAME="Chang" JOB="Manager"
/>
  <TITLE ID_NO="2" FIRST_NAME="Hook" LAST_NAME="Hu"
JOB="SoftwareEngineer" />
  <TITLE ID_NO="3" FIRST_NAME="Jackie" LAST_NAME="Yu"
JOB="SoftwareEngineer" />
  <TITLE ID_NO="4" FIRST_NAME="Jerry" LAST_NAME="Liu" JOB="Manager" />
</EMPLOYEE>
```

関連プロシージャ

XMLIMPORT

5.15 XMLIMPORT

目的

データをXMLからDBMasterにインポートします。

原型

```
XMLIMPORT(
```

```
        VARCHAR(256)    FILE_PATH,
        VARCHAR(16000   OBJECT_STR,
        VARCHAR(256)    OPTION_STR,
        VARCHAR(256)    LOG_PATH)
```

説明

XMLIMPORTシステム・ストアド・プロシージャは、データをXMLからDBMasterにインポートすることができるプログラム可能なインターフェースです。SYSADMとDBAのみが、このストアド・プロシージャを呼び出すことができます。又、XMLIMPORTはシステム・ストアド・プロシージャですので、他のユーザーに実行権限を与えることはできません。

XMLIMPORTは、表をXMLファイルからDBMasterの表にインポートします。XMLファイルからインポートする際に、XMLファイル全体を解読する（ファイル内容の分析と、データを表にインポート）代わりに、データベースに保存するだけです。インポートされるXMLファイルはサーバー側に、XMLファイルのインポートの際に生成されるログファイルは、クライアント機に保存されます。

XMLファイル全体を解読する代わりに単に保存したいだけであれば、XMLファイルを保存するために「key」を指定する必要があります。そうすることで、保存したXMLファイル用にデータベースに問い合せる際に、キー値を使うことができます。

引数名	データ型	サイズ（バイト）	解説	大文字と小文字の区別
file_path	varchar	256	エクスポートするXMLファイルの絶対パス	オペレーティング・システムに依ります
object_str	varchar	16000	エクスポートするオブジェクトのディスクリプション文字列	XMLのタグは、大文字と小文字を識別します。表名と表のカラム名は、DBMasterの設定に依ります

引数名	データ型	サイズ(バイト)	解説	大文字と小文字の区別
option_flag	varchar	256	オプション・フラグのディスクリプション文字列	区別しません
log_path	varchar	256	クライアント側のエラーログの絶対パス	オペレーティング・システムに拠ります

図5-3 XMLIMPORTの引数

XMLIMPORTに引数を与える

データベースからインポートするXMLファイルは、サーバー側に生成されます。最初の文字列file_pathには、絶対パスを指定します。

2つ目のobject_str引数は、インポートするオブジェクトを指定します。この情報には、ドキュメントのレベル、カスタマイズしたカラムのタグ名間のマップ、カスタマイズした表のタグ名とデータベースの表名の間のマッピング同様に、挿入した表のカラム名が含まれます。フォーマットは、以下のとおりです。

```
object_str =:
       { <table_element> [; <table_element>]...}


<table_element> =
       { <document mapping information>#<table mapping information> }


<document mapping information> =:
       {<document level string>[(<column tag names>)]


<document level string> =: {/<level1> [/<level2>/.....]}


<column tag names> =: {<tag1> [, <tag2>]...}
```

```
<table mapping information> =: <table import definition>

<table import definition> =: { <insert sql statement> | <target table
name>[(<table column names>)] }

<insert sql statement> =: INSERT INTO <target table name> [(<table column
names>)] VALUES (<value list>)

<table column names> =: {<col1> [, <col2>] ...}

<value list> =: {<insert value>, <insert value>,...}

<insert value> =: {<constant> | <expression>}
```

図5-4 サンプルXMLファイル

XMLファイルを解読し、内容を表に保存する代わりにファイル全体を保存したい場合、<column tag names>に特別なハンドリングを使います。

<table_element>は、表を表します。<element>間に使用するデリミタは、セミコロン(;)です。

<document level string>には、ルート・レベルから表レベルまでのドキュメントのレベルを指定します。

```
<root>
   <database>
      <table1>
          <column1>
          </column1>
          <column2>
          </column2>
      </table1>
      <table2>
```

```
      </table2>
   </database>
</root>
```

図5-5 サンプルXMLファイル

図5.2のサンプルXMLファイルに基づき、<database>の<table1>タグに保存したデータをインポートするために、「/root/database/table1」の<document level string>を指定します。

<column tag names>には、表に挿入しようとするカラムのタグを指定します。<column tag names>を定義しない場合、特定の表のタグにあるカラムのタグ全てが挿入されます。

<table import definition>には、<insert sql statement>或いはTABLE_NAME [<table column names>]のフォーマットのいずれかを使用します。<insert sql statement>を使用すると、挿入されるSQL文は、次のようになります。

```
INSERT INTO <target table name> [(<table column names>)] VALUES (<value list>)
```

挿入されるカラムには、<table column names>を指定します。<table column names>を指定しない場合、ユーザーが目的の表に全てのカラムを挿入しようとしていることを意味します（これは、一般のINSERT SQL文の構文と同じです）。また、<column tag names>が<document mapping information>に存在する場合、<column tag names>で指定するカラム・タグの数は、<value list>のホスト変数の数と同等にする必要があります。<document mapping information>に存在する<column tag names>が無い場合、元の要素にあるカラム・タグの全てが、目的の表に挿入されることを意味します。DTDファイルのスキーマ情報は、タグの数が<value list>にあるホスト変数の数と等しいかどうかをチェックするためにも使用されます。

<document mapping information>ファイルにある<table column names>、<value list>、<column tag names>間のマッピングは、適切に行わなければなりません。<column tag names>は、<value list>ファイルのホスト変数にマップされます。<value list>のシーケンス値と合わせる<table column names>のカラム・シーケンスと、タグ<column tag names>のシーケンスは、どの値を<value list>に挿入するかを決定します。

<target table name>[(<table column names>)]を使用する場合、<target table name>に挿入する表を指定します。この<target table name>は、<document level string>の最後のレベルにマップされます。

このフォーマットを使用すると、定数値の挿入や式の挿入は使用できません。<document mapping information>に<column tag names>を指定しない場合、<table column names>も存在しません。<document mapping information>に<column tag names>がある場合、<column tag names>にあるタグの数は、<table column names>にあるカラム数と同等にする必要があります。

<table column names>には、挿入するマップされた表のカラムを指定します。<table column names>を指定しない場合、表のカラムは全て挿入されます。この場合、<document mapping information>には<column tag names>がありません。DTDファイルのスキーマ情報は、元の要素にあるタグの数と目的の表にあるカラムの数が等しいかどうかをチェックします。

ユーザーは、<table column names>と<column tag names>間のマッピングを行います。<column tag names>にあるタグの位置は、<table column names>にあるカラムの位置にマップしなければなりません。

使用例

➲ 使用例1

<table column names>が(c1, c2, c3)、<value list>が(?,?,?)、<column tag names>が(tg1, tg2, tg3)の場合、tg1の値はc1に挿入され、tg2の値はc2に挿入され、tg3の値はc3に挿入されます。

➲ 使用例2

表t1に4つのカラム（c1、c2、c3、c4）があり、インポートしようとしているXMLの要素に4つのタグ（tg1、tg2、tg3、tg4）があると想定します。更に、obj_strは、「/root/book/order(tg1, tg2)#insert into t1 (c1, c2, c3) values (?,?+3, 5)」です。文字列から、表t1を目的の表とすることを決定し、表t1のカラムc1にはタグtg1の挿入された値があります。カラムc2にはタグtg2とtg3の挿入された値があり、カラムc3には挿入された定数5があると想定します。

➲ 使用例3

<document mapping information>に<column tag names>ファイルの使用を指定しない場合、XMLカラムのタグのシーケンスが<table column names>にあるもののシーケンスと合致し、基の要素にあるカラムのタグ全てが目的の表に挿入されることを表します。

目的の表t2には5つのカラム（c1、c2、c3、c4、c5）があると想定します。更に、XMLファイルのタグのシーケンスは、tg1、tg2、tg3、tg4とします。obj_strが「/root/book/order#insert into t2 (c1, c2, c3, c4, c5) values (?, ?, ?, ?, 6)」の場合、t2のc1にtg1の値が挿入され、t2のc2にtg2の値が挿入され、t2のc3にtg3の値が挿入され、t2のc4にtg4の値が挿入され、t2のc5に定数6が挿入されます。

obj_strが「/root/book/order(tg1, tg2, tg3, tag4)#insert into t1 values (?, ?, ?, ?)」の場合、目的の表の全カラムに4つのタグを挿入しようとすることを意味します。t1のc1にtg1の値が挿入され、t1のc2にtg2の値が挿入され、t1のc3にtg3の値が挿入され、t1のc4にtg4の値が挿入されます。

➲ 使用例4

obj_strが「/root/book/order(tg1, tg2)#insert into t1 values (?, ?, acos(1))」の場合、t1のc3にはacos(1)の結果が挿入されます。

➲ 使用例5

XMLファイル全体を解読し、内容を保存する（例えば、XMLファイル全体を解読し、表にXMLファイルのデータを保存する）代わりに、レコードにXMLファイル全体を保存しようとするユーザーは、<column tag names>に「仮想タグ」を指定する必要があります。この特殊な「仮想タグ」は、「_XML_FILE_」と名づけられます。

この「_XML_FILE_」がカラムのタグ名のように使用される場合、この特殊な「仮想タグ」を通じてカラム・タグによって表されるカラムは、キー値として使用されます。加えて、<value list>ファイルにマップされた値は、更なる計算の無い1つのホスト変数になります。

次のオブジェクト文字列「/root/book/order(tag1, tag2, XML_FILE_)#insert into t2 (c1, c2, c3, c4, c5) values (?+2, ?*5, ?, 7, 8)」が使用される場合、ファイル全体は表t2のc3に挿入されます。

オブジェクト文字列の<table_element>が、「/root/book/order(tag1, tag2, _XML_FILE_)#customer(firstname, lastname, xml_file)」が表「customer」に使用される場合、firstnameがtag1からXMLファイルに挿入されます。加えて、lastnameはXMLファイルにtag2から挿入されます。xml_fileがXMLファイル全体から挿入されます。Firstnameとlastnameが特殊なXMLファイル見つけるためのキーとして使われます。

➲ 使用例6

<column tag names> = <tag1, tag2, tag3>と<table column names> = <c1, c2, c3>には、3つのマップがあります。tag1 <-> c1、tag2 <-> c2、tag3 <-> c3です。タグ名とカラム名は、両方あるか、全くないかのいずれかです。(tag1, ,tag3)のような空のタグ名や、空のカラム名は、認められていないことを意味します。カスタマイズされたタグ名の全てが指定されているか、全く指定されていないかのどちらかです。

つまり、オブジェクト文字列「/root/book/order(tag1, , tag2)#insert into t2 (c1, c2) values (?, ?, ?)」は認められません。オブジェクト文字列「/root/book/order(tag1, tag2, tag3)#insert into t2 (c1, c2, c3, c4) values (?, ?, ?,)」は認められます。t2のc4に何が挿入されるかは、表のスキーマ情報に拠ります。

3つ目に、option_flag文字列は大文字と小文字を区別しません。option_flag文字列が設定される場合、インポートされるXMLファイルのcolumn_as_attributeカラムは、属性として扱われます。option_flag文字列が設定されない場合、カラムはXMLファイルの要素として扱われます。

```
Option_flag=:{[<attribute>[;<attribute>]…]}
<attribute>=:
{
column_as_attribute
}
```

最後に、XMLファイルのインポートの際に生成されたエラーのログファイルは、クライアント機のlog_pathに保存されます。

XMLファイルをインポートする

/usr/johnディレクトリにXMLファイルxmlimport.xmlがあると想定します。ファイルは以下のようになります。

```
<ROOT>
   <EMPLOYEE>
     <TITLE>
       <TAG1>1</TAG1>
       <TAG2>Eddie</TAG2>
       <TAG3>Chang</TAG3>
       <TAG4>Manager</TAG4>
     </TITLE>
     <TITLE>
       <TAG1>2</TAG1>
       <TAG2>Hook</TAG2>
       <TAG3>Hu</TAG3>
       <TAG4>SoftwareEngineer</TAG4>
     </TITLE>
     <TITLE>
       <TAG1>3</TAG1>
       <TAG2>Jackie</TAG2>
       <TAG3>Yu</TAG3>
       <TAG4>SoftwareEngineer</TAG4>
     </TITLE>
     <TITLE>
       <TAG1>4</TAG1>
       <TAG2>Jerry</TAG2>
       <TAG3>Liu</TAG3>
```

```
      <TAG4>Manager</TAG4>
    </TITLE>
    <NUMBER>
      <NO>1</NO>
      <FIRST_NAME>Eddie</FIRST_NAME>
      <LAST_NAME>Chang</LAST_NAME>
      <PHONE>2145678</PHONE>
    </NUMBER>
    <NUMBER>
      <NO>2</NO>
      <FIRST_NAME>Hook</FIRST_NAME>
      <LAST_NAME>Hu</LAST_NAME>
      <PHONE>2335678</PHONE>
    </NUMBER>
    <NUMBER>
      <NO>3</NO>
      <FIRST_NAME>Jackie</FIRST_NAME>
      <LAST_NAME>Yu</LAST_NAME>
      <PHONE>2346678</PHONE>
    </NUMBER>
    <NUMBER>
      <NO>4</NO>
      <FIRST_NAME>Jerry</FIRST_NAME>
      <LAST_NAME>Liu</LAST_NAME>
      <PHONE>2345671</PHONE>
    </NUMBER>
  </EMPLOYEE>
<ROOT>
```

次のデータベースのスキーマにimportxml.xmlファイルに記録されたデータをインポートします。

```
Database Name: DB_TEST

Table Name: TB_CARD(ID CHAR(30), FNAME CHAR(30), LNAME CHAR(30), WORK
CHAR(30))

Table Name: TB_CONTACT(NO CHAR(30), FIRST_NAME CHAR(30), LAST_NAME CHAR(30),
PHONE CHAR(30))
```

上記の.xmlファイルの内容から、<EMPLOYEE>要素の下には、2つの副要素があることがわかります。データベース・レベルとして<EMPLOYEE>の要素をマップし、表レベルとして<TITLE>を、他の表レベルとして<NUMBER>をインポートするデータベースにマップすることができます。

CARD表に<TITLE>を、CONTACT表に<NUMBER>をインポートすると想定します。データベースの表とXML文書のタグのマップは、次のとおりです。

```
/ROOT/EMPLOYEE/TITLE -> / DB_TEST / TB_ CARD

/ROOT/EMPLOYEE/NUMBER -> / DB_TEST / TB_ CONTACT
```

XML文書のタグと表のカラムの間のマップは、次のとおりです。

/ROOT/EMPLOYEE/TITLEの下にある要素（<TITLE>とTB_ CARD表の間のマップ）：

```
TAG1 -> NO

TAG2 -> FIRST_NAME

TAG3 -> LAST_NAME

TAG4 -> JOB
```

/ROOT/EMPLOYEE/NUMBERの下にある要素（<NUMBER>とTB CONTACT表の間のマップ）：

```
NO -> NO

FIRST_NAME -> FIRST_NAME

LAST_NAME -> LAST_NAME

PHONE -> PHONE
```

加えて、xmlimport.xmlではカラムが目的のXMLファイルの要素として扱われていることがわかります。最後に、ログファイルが/client/john/xmlimport.logであると想定します。

表TB_ CARDにインポートする場合、/ROOT/EMPLOYEE/TITLEにある要素がインポートされます。TAG1はカラムIDにマップされ、TAG2はカラムFNAMEにマップされ、TAG3はカラムLNAMEにマップされ、TAG4はカラムWORKにマップされます。

表TB_CONTACTをインポートする場合、/ROOT/EMPLOYEE/NUMBERにある要素がインポートされます。<NUMBER> タグにある全ての要素がインポートされ、表CONTACTに直接マップすると想定されます。

XMLのタグは大文字と小文字を区別することに留意して下さい。つまり、この例のROOT、EMPLOYEE、TITLE、TAG1、TAG2、TAG3は、大文字になります。表名と表のカラム名の大文字と小文字は、DBMasterの設定に拠ります。

➲ **上記のファイルでXMLIMPORTを使う:**

1. ファイルは常にサーバー側にあるので、指定する絶対パスもサーバー側にします。引数file_pathには、「/usr/john/xmlimport.xml」を指定します。

2. object_strには、次のように指定します。

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#INSERT INTO TB_CARD
(ID,FNAME,LNAME,WORK) VALUES (?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER#
tb_contact'
```

或いは次のように指定します。

```
'/ROOT/EMPLOYEE/TITLE#INSERT INTO TB_CARD (ID,FNAME,LNAME,WORK) VALUES
(?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER# tb_contact'
```

或いは次のように指定します。

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#CARD
(C1,C2,C3,C4);/ROOT/EMPLOYEE/NUMBER#contact'
```

3. 使用するオブジェクト文字列には、複数のフォーマットがあります。

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#INSERT INTO TB_CARD
(ID,FNAME,LNAME,WORK) VALUES (?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER#
TB_CONTACT'
```

或いは、4つのタグが4つのカラムにマッピングしているので、タグのシーケンスはカラムと同じになります。

```
'/ROOT/EMPLOYEE/TITLE#INSERT INTO TB_CARD (ID,FNAME,LNAME,WORK) VALUES
(?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER# TB_CONTACT'
```

或いは、次のように指定します。

```
'/ROOT/EMPLOYEE/TITLE#INSERT INTO TB_CARD VALUES
(?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER# TB_CONTACT'
```

或いは、ホスト変数で更なる計算が必要ない場合、次のようになります。

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#
TB_CARD(ID,FNAME,LNAME,WORK);/ROOT/EMPLOYEE/NUMBER# TB_CONTACT'
```

4. カラムはXMLファイルの要素として扱われるので、ここではoption_flagをセットしません。これらのカラムが要素として扱われない場合、option_flagがセットされます。

```
option_flag =: {[<attribute> [;<attribute>]...]}

<attribute> =:

{

column_as_attribute

}
```

5. log_pathは、「/client/john/xmlimport.log」になります。これはXMLIMPORTの処理の際にエラーが記録された場所になります。

6. obj_strのあるフォームを使ってXMLIMPORTを呼び出します。

```
call  XMLImport (

'/usr/john/xmlimport.xml',
```

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#
TB_CARD(ID,FNAME,LNAME,WORK);/ROOT/EMPLOYEE/NUMBER# tb_contact',
'',
'/client/john/xmlimport.log');
```

関連プロシージャ

XMLEXPORT

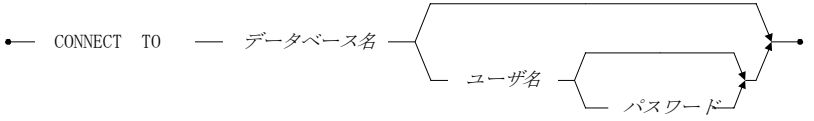
6 dmSQL文

6.1 CONNECT

目的

データベースに接続します。

構文



データベース名	接続するデータベースの名前
ユーザー名	データベースに接続するユーザーの名前
パスワード	ユーザーのパスワード

説明

CONNECT文は、データベースとの接続を確立します。データベース名は大文字と小文字を区別しません。CONNECT以上のセキュリティ権限をもつユーザーなら誰でもCONNECTを実行することができます。

データベースに接続する前に、コンピュータの**dmconfig.ini**ファイルに目的のデータベースのデータベース環境設定セクションが存在することを確認する必要があります。データベースがローカル・コンピュータで作成された場合は、データベースの環境設定セクションは既に存在するはずです。データベースがリモート・コンピュータで作成された場合は、データベースの環境設定セクションを追加します。

シングルユーザー・データベースの接続にCONNECTを使用すると、データベース接続を確立すると共にデータベースを起動します。シングルユーザー・データベースには、唯一人のユーザーだけが接続することができます。

シングルユーザー・データベースに接続する前に、**dmconfig.ini**ファイルのDB_DBDIRキーワードにデータベース・ディレクトリを指定しておきます。

クライアント／サーバー・データベースと接続するには、データベース・サーバーを起動しているときにCONNECTを使用します。データベース・サーバーを起動していないときは、接続する前にサーバーを起動させます。

クライアント／サーバー・データベースは、サーバーが起動するホストコンピュータのIPアドレスとデータベースのポート番号を指定しておきます。IPアドレスとポート番号は、**dmconfig.ini**ファイルのDB_SVADRとDB_PTNUMキーワードで指定します。DB_SVADRキーワードには、IPアドレスの代わりにホスト名を指定することもできます。

クライアント／サーバー・データベースとの接続は、接続タイムアウトが発生するまで試みられます。接続タイムアウトは、**dmconfig.ini** ファイル

のDB_CTIMOキーワードで指定します。DB_CTIMOキーワードは、シングルユーザー・データベースには適用されません。

ユーザー名とパスワードはオプションではありません。唯一省略できる例外はパスワードがNULLのときです。**dmconfig.ini**ファイルのDB_USRIDとDB_PASWDキーワードを使用すると、CONNECT文のユーザー名とパスワードを省略することができます。DB_USRIDキーワードは初期設定のユーザー名を指定し、DB_PASWDキーワードは初期設定のパスワードを指定します。DBMasterはユーザー名とパスワードを常に同じ所からもってきます。一方をCONNECT文で指定し他方を設定ファイルで指定することはできません。CONNECT文でユーザー名とパスワードを与えたときは、DB_USRIDとDB_PASWDキーワードで指定された値は無視されます。

使用例

➲ **参照値**1　**dmconfig.ini**ファイルの**TUTOR1**データベース・セクションに**DB_DBDIR**キーワードの値を設定します：

```
[TUTOR1]
DB_DBDIR = C:\DBMASTER\DATABASE\TUTOR1
```

➲ **使用例1**　ユーザー**jenny**がパスワード**grala833**を指定してシングルユーザー・データベース**Tutor1**に接続します：

```
CONNECT TO Tutor1 jenny grala833;
```

➲ **参照値2a**　**dmconfig.ini**ファイルの**TUTOR2**データベース・セクションに**DB_SVADR**と**DB_PTNUM**キーワードの値を設定します：

```
[TUTOR2]
DB_SVADR = 192.72.116.137
DB_PTNUM = 35400
```

➲ **参照値2b**　IPアドレスの代わりにホスト名を**DB_SVADR**に設定します：

```
[TUTOR2]
DB_SVADR = mars.syscom.com.tw
```

```
DB_PTNUM = 35400
```

➲ **使用例2**　ユーザー**amanda**がパスワード**grixa944**を指定してマルチユーザー・データベース**Tutor2**に接続します：

```
CONNECT TO Tutor2 amanda grixa944;
```

➲ **参照値3**　dmconfig.iniファイルのTUTOR2セクションにDB_SVADR、DB_PTNUM、DB_USRID、DB_PASWDキーワードの値を設定します：

```
[TUTOR2]
DB_SVADR = 192.72.116.137
DB_PTNUM = 35400
DB_USRID = vivian
DB_PASWD = shuka828
```

DB_SVADRは、参照値2bと同様、IPアドレスの代わりにホスト名を与えることもできます。

➲ **使用例3**　ユーザー**vivian**がパスワード**shuka828**を使用してマルチユーザー・データベース**Tutor2**に接続します。ユーザー名とパスワードは**dmconfig.ini**の**DB_USRID**と**DB_PASWD**キーワードで指定されているのでSQL文では指定しません。ユーザー名とパスワードをSQL文で与えると、**DB_USRID**と**DB_PASWD**キーワードで指定されている値は無視されます：

```
CONNECT TO Tutor2;
```

関連dmSQL

DISCONNECT

GRANT (セキュリティ権限)

REVOKE (セキュリティ権限)

START DATABASE

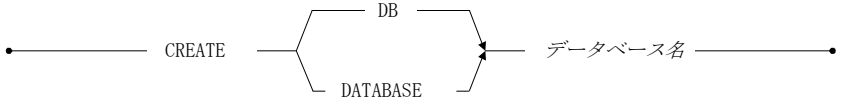
TERMINATE DATABASE

6.2 CREATE DATABASE

目的

データベースを新規に作成します。

構文



データベース名　　新規に作成するデータベースの名前

説明

CREATE DATABASE文は、データベースを新規作成します。データベースを作成するには、データベース・ディレクトリのオペレーティング・システム書き込み許可がDBMasterに与えられていなければなりません。CREATE DATABASEは、任意のユーザーが実行することができます。

DBMasterは、データベースの全ての環境設定情報を**dmconfig.ini**ファイルに格納します。このファイルには、コンピュータから接続できるデータベース毎に構成セクションがあります。**dmconfig.ini**ファイルはASCIIテキストファイルであり、テキストエディタで編集することができます。

各データベース環境設定は、セクション見出しと一連のキーワード行から構成されます。セクション見出し行は、[データベース名]の形式です。キーワード行は、キーワードと対応する値で構成されます。一つのキーワードに複数の値を指定する場合は、各々の値をスペースまたはコンマで区切ります。キーワードの値は、その目的によりデータベースの起動時またはユーザーの接続時に使用されます。

dmconfig.iniファイルのキーワードは、大文字と小文字を区別しません。キーワードの値は、データベースが作動するオペレーティング・システムによっては大文字と小文字を区別することがあります。データベースの作成時には、**dmconfig.ini**ファイルのデータベース・セクションが調べられます。データベースと同じ名前のデータベース・セクションがあれば、セクション内に指定されている値を使用してデータベースを作成します。データベースと同じ名前のデータベース・セクションが無ければ、初期設定値を使用してデータベースを作成し、データベース・セクションを新規に追加します。

データベースの名前は、接続できる全てのコンピュータ内で一意にします。**dmconfig.ini**にはローカルおよびリモートの全てのデータベース設定情報が含まれるので、同じデータベース名があると競合してしまいます。データベース名の最大長は32字です。文字、数字、アンダースコアを使用することができます。データベース名は、大文字と小文字を区別しません。

DBMasterの物理ストレージモデルでは、データを格納する物理ストレージはファイルです。ファイルはオペレーティング・システムが管理し、一方、ファイル内のデータはDBMSが管理します。DBMasterは、データ、BLOB、ジャーナルの3種類のファイルを使用します。

データファイルとBLOBファイルは、ユーザーとシステムのデータを格納します。データとBLOBファイルは類似の特性をもっていますが、パフォーマンスをよくするために別々の方法で管理します。データファイルは表と索引のデータを格納し、一方、BLOBファイルはバイナリラージオブジェクト(BLOB)を格納します。

ジャーナル・ファイルは、データベースの全ての変更履歴と変更ステータスをリアルタイムに記録する特殊なファイルです。ジャーナルは、失敗したトランザクションによる変更のundo、成功したけれどもデータベース損傷によってディスクに書き込まれていない変更のredoを可能にします。ジャーナル・ファイルはデータベース管理システムによってのみ使用されます。ユーザーデータの格納には使用されません。

DBMasterの論理ストレージモデルでは、データベース情報を管理可能なエリアに区分けするのに用いられる論理ストレージ構造は表領域です。表領

域には、種々の表や索引が含まれます。表領域内のデータはDBMSによって管理されますが、物理的にはファイルに格納されます。3種類、標準、自動拡張、システムの表領域があります。

標準表領域には、最低1つのデータファイルあるいはBLOBファイルがあります。標準表領域は固定サイズであり、サイズを拡大するには、既存ファイルを拡大するか新規ファイルを追加します。標準表領域は最大32767個のファイルをもつことができ、ファイルサイズの合計は最大8TBです。Unixプラットフォームでは、ローデバイスに標準表領域を置くことができます。

注：　ローデバイスの詳細情報はUnixシステムのマニュアルを参照してください。

自動拡張表領域は、必要に応じて最大8TBまで自動的にサイズを拡大します。自動拡張表領域は、1個のデータファイルをもたなければならず、1個のBLOBファイルをもたせることができます。自動拡張表領域にファイルを追加するには、最初に標準表領域に変更します。データファイルだけの自動拡張表領域を作成したときは、後にBLOBファイルを追加することができます。自動拡張表領域はローデバイスをサポートしません。

システム表領域は、データベース作成時にDBMasterによって生成されます。各データベースにはシステム表領域が一つあり、スキーマ、セキュリティ、ステータス情報のシステムカタログ表が格納されます。システム表領域は、Unixローデバイス上に作成する場合を除いて、自動拡張表領域として作成されます。システム表領域を標準表領域に変換することもできます。システム表領域は、初期サイズ600KBのデータファイル、初期サイズ20KBのBLOBファイルで作成されます。

DBMasterは、システムデータファイルとシステムBLOBファイルを1個ずつシステム表領域に生成し、ユーザーデータファイルとユーザーBLOBファイルを1個ずつ初期設定のユーザー表領域に作成します。この他に、データベーストランザクションのログを取るために、少なくとも1個のシステムジャーナル・ファイルも生成します。

システムファイルの初期設定名は、DATABASE.SDB、DATABASE.SBB、DATABASE.JNLです。DATABASEはデータベース名です。**dmconfig.ini**ファイルのDB_DBFIL、DB_BBFIL、DB_JNFILキーワードを使用して初期設定名を変更することができます。DB_DBFILはシステム・データファイル名、DB_BBFILはシステムBLOBファイル名、DB_JNFILはシステム・ジャーナル・ファイル名を指定します。データベース作成前にファイル名を指定しておかないと、初期設定名が使用されます。データベース作成後にシステムファイル名を変更することはできません。

ユーザーファイルの初期設定名は、DATABASE.DBとDATABASE.BBです。DATABASEはデータベースの名前です。**dmconfig.ini**ファイルにあるDB_USRDBとDB_USRBBキーワードを使用して初期設定名を変更することができます。DB_USRDBはユーザーデータファイル名とサイズを指定し、DB_USRBBはユーザーBLOBファイル名とサイズを指定します。この二つのキーワードを使用するときは、データページ数またはBLOBフレーム数でファイルサイズを指定します。ファイル名とファイルサイズはスペースまたはコンマで区切ります。初期設定のユーザーファイル名を変更するときは、データベース作成前にファイル名を指定しておきます。

ジャーナル・ファイルは、最大8個まで使用することができます。マルチジャーナル・ファイルを作成するには、DB_JNFILキーワードの後ろに、スペースまたはコンマで区切って、複数のファイル名を追記します。DBMasterは、データベース作成時に自動的にこれらのジャーナル・ファイルを作成します。データベース作成後にジャーナル・ファイルを追加することもできます。ジャーナル・ファイルを追加するには、ジャーナル・ファイル名を追記し新ジャーナルモードでデータベースを再起動します。

ファイル名にパスを含めるときは、Windows 95とWindowsNTではドライブと絶対パスを指定します。Unixでは絶対パスまたは相対パスを指定します。パスを含まないファイルは、**dmconfig.ini**ファイルのDB_DBDIRキーワードで指定されたディレクトリに、DB_DBDIRキーワードが無いときはアプリケーションディレクトリに作成されます。ファイル名の最大長は255字です。オペレーティング・システムが許すスペース以外の任意の文字を使用することができます。

システムファイルの初期設定サイズは、データファイルが600KB、BLOBファイルが20KB、ジャーナル・ファイルが4000KBです。初期設定のファイルサイズを変更するには、**dmconfig.ini**ファイルのDB_BFRSZ、DB_JNLSZキーワードを使用します。

DB_BFRSZキーワードは、BLOBファイルのフレームサイズを指定し、同時にシステムBLOBファイルのサイズも変更します。初期設定サイズを使用したくないときは、データベース作成時にDB_BFRSZを与えておきます。データベース作成後に変更することはできません。

DB_JNLSZキーワードは、システムジャーナル・ファイルのサイズをジャーナル・ブロック数で指定します。ジャーナル・ブロックは、ジャーナル・ファイルの基本記憶単位です。ジャーナル・ブロックのサイズは**dmconfig.in**iファイルのDB_PGSIZの値 から決定されます。データベース上で実行した全てのトランザクションのレコードが格納されます。各ジャーナル・ブロックは、できる限りのトランザクション情報をブロックに格納します。DB_JNLSZキーワードの値は、23～524287ブロック数の範囲で設定します。キロバイトに換算するには**dmconfig.in**iファイルのDB_PGSIZの値 を乗算します。。複数のジャーナル・ファイルをもたせる場合、各ジャーナル・ファイルはDB_JNLSZで指定したサイズで作成されます。DB_JNLSZの初期設定値は1000です。DB_JNLSZキーワードは何時でも変更することができます。ただし、次に、新ジャーナルモードでデータベースを起動するまでは有効になりません。

ユーザーファイルの初期設定サイズは、データファイルが600KB、BLOBファイルが20KBです。初期設定サイズを変更するには、**dmconfig.ini**ファイルのDB_USRDBとDB_USRBBキーワードを使用します。

DB_USRDBキーワードは、データファイルのサイズをデータページ数で指定します。データページは**dmconfig.in**iファイルにDB_PGSIZの値から決定されます、、表レコード、索引キー、ページ内に収まる小さいBLOBデータ等を格納します。各データページは、できる限りの表行、索引キーをページ内に格納します。DB_USRDBキーワードの値は2～524287ページの範囲で設定します。キロバイトに換算するにはこの値に**dmconfig.in**iファイルのDB_PGSIZの値 を乗算します。 DB_USRDBの初期設定値は150です。

DB_USRBBキーワードは、BLOBファイルのサイズをBLOBフレーム数で指定します。BLOBフレームはBLOBファイルの基本記憶単位です。BLOBフレームは、バイナリラージオブジェクト、グラフィックス、オーディオとビデオ、あるいはデータページに適合しないラージテキスト等を格納します。1つのBLOBフレームには、1つのBLOBデータしか格納しません。BLOBフレームのサイズは8KB～256KBの範囲でDB_BFRSZキーワードで指定します。DB_USRBBキーワードの値は2～524287フレーム数の範囲で指定します。この値にDB_BFRSZの値を乗算して、実際のファイルサイズ(KB)を計算します。DB_USRBBの初期設定値は2です。

DBMasterには、4レベルのセキュリティ権限、CONNECT、RESOURCE、DBA、SYSADMがあります。

CONNECTセキュリティ権限は、それを有するユーザーが、データベースに接続し、システム表を閲覧し、オブジェクトの所有者、DBA、SYSADMによって権限が与えられたオブジェクトをアクセスすることを可能になります。CONNECTセキュリティ権限は、データベース・オブジェクトを新規作成すること認めていません。他の権限を与える前に、CONNECTセキュリティ権限を与える必要があります。

RESOURCEセキュリティ権限は、表、索引、ビュー、シノニム、ドメインを作成/削除することを認可します。ユーザーは、自分が作成した表、ビュー、シノニム、ドメインだけを削除することができます。また、自分が作成したデータベース・オブジェクトのオブジェクト権限を他のユーザーに与えたり、取り消すことができます。RESOURCEセキュリティ権限をもつユーザーは、同時にCONNECTセキュリティ権限ももちます。

DBAセキュリティ権限は、データベースを起動、終了、バックアップし、データベース・リソース、表領域、ファイルを管理し、全ての表、索引、ビュー、シノニム、ドメインをオブジェクト権限なしにアクセスすることを認可します。また、他のユーザーが所有する任意のデータベース・オブジェクトのオブジェクト権限を与え、変更し、取り消すことができます。DBAは、新規ユーザーにセキュリティ権限を与えたり、新規グループを作成することはできませんが、既存グループにユーザーを追加/削除すること

はできます。DBAセキュリティ権限をもつユーザーは、同時にRESOURCEとCONNECTセキュリティ権限ももちます。

SYSADMセキュリティ権限は、セキュリティ権限をユーザーに与え取り消す、グループを作成し削除する、グループにユーザーを追加し削除することが認可されます。他のユーザーのパスワードを変更することも許されます。SYSADMセキュリティ権限をもつユーザーは、データベースに一人だけです。DBMasterは、データベース作成時に、ユーザー名SYSADMを付けて自動的にこのユーザーを作成します。SYSADMは、SYSADMセキュリティ権限を他のユーザーに与えることはできません。SYSADMは、同時にDBA、RESOURCE、CONNECTセキュリティ権限ももちます。

CREATE DATABASE文を実行すると、データベースが新規作成され、起動し、SYSADMユーザーとしてデータベースに接続されます。この時点では、パスワードはSYSADMユーザーに割当てられていません。作成直後のデータベースはシングル・ユーザーモードで起動しており、他のユーザーはデータベースにログオンすることはできません。SYSADMのパスワードを変更して無認可のデータベースアクセスを防ぐようにします。新パスワードを有効にして他のユーザーが接続できるようにするには、データベースを終了し、シングルモードまたはマルチユーザー・モードで再起動します。

初期設定では、データベースはシングルユーザー・モードで起動します。マルチユーザー・モードで起動させるには、クライアント側の**dmconfig.ini**ファイルでDB_SVADRとDB_PTNUMキーワードを設定し、サーバー側の**dmconfig.ini**ファイルでDB_PTNUMキーワードを設定します。

DB_SVADRキーワードは、サーバーが起動するコンピュータのIPアドレスまたはホスト名を指定します。このキーワードは、クライアント側でのみ必要であり、サーバー側ではオプションです。IPアドレスまたはホスト名を指定するには、DB_SVADRキーワードの値を有効なIPアドレスまたはホスト名に設定します。ホスト名を使用するときは、ドメインネームサービス（DNS）がコンピュータに適切に設定されていることを確認します。

DB_PTNUMキーワードは、サーバーにバインドされるポート番号を指定します。このキーワードは、クライアントとサーバーの両方に必要です。ポ

ート番号を指定するには、DB_PTNUMキーワードに1025～65535の範囲の値に設定します。ポート番号を指定しないと、初期設定の番号23000が使用されます。

使用例

- **使用例1**　初期設定パラメータを用いて**Accounts**シングルユーザー・データベースを新規作成します。この時点で**dmconfig.ini**ファイルにはデータベース・セクションは作られていません。初期設定の**ACCOUNTS.SDB**、**ACCOUNTS.SBB**、**ACCOUNTS.DB**、**ACCOUNTS.BB**、**ACCOUNTS.JNL** ファイルが初期設定サイズ600KB (**.SDB, .DB)**、20KB (**.SBB, .BB)**、4000KB **(.JNL)** でアプリケーション・ディレクトリに作成されます。作成したデータベースをマルチユーザー・モードで起動させるには、**dmconfig.ini**ファイルの**Accounts**データベース構成セクションに**DB_SVADR**と**DB_PTNUM**キーワードを追加します：

```
CREATE DATABASE Accounts;
```

- **使用例2**　下の引用例にある**dmconfig.ini**の設定を使用して**Accounts**データベースを新規作成します：

```
CREATE DATABASE Accounts;
```

- **引用例**　以下のデータベース・セクションが**dmconfig.ini**ファイルに設定されています。シングルユーザー・データベースを **C:\DATABASE\ACCOUNTS**ディレクトリにセキュリティ付きで作成します。データベース・ファイル名は、システム・データファイル **ACCOUNTS.SDB**、システムBLOBファイル**ACCOUNTS.SBB**、ユーザーデータファイル**ACNTDATA.DB**、ユーザーBLOBファイル**ACNTBLOB.BB**、ジャーナル・ファイルは**ACNTHIST.JN1**、**ACNTHIST.JN2**、**ACNTHIST.JN3**の3個です。ファイルサイズは、システム・データファイル600KB、システムBLOBファイル20KB、ユーザー・データファイル1000KB、ユーザーBLOBファイル8000KB、ジャーナル・ファイル2000KBです。作成したデータベースをマルチユーザー・モードで起動させるに

は、**DB_SVADR**と**DB_PTNUM**キーワードを**dmconfig.ini**ファイルの**Accounts**データベース構成セクションに追加します：

```
[ACCOUNTS]
DB_DBDIR = C:\DATABASE\ACCOUNTS
DB_DBFIL = ACCOUNTS.SDB
DB_BBFIL = ACCOUNTS.SBB
DB_USRDB = ACNTDATA.DB 250
DB_USRBB = ACNTBLOB.BB 250
DB_BFRSZ = 32
DB_JNFIL = ACNTHIST.JN1, ACNTHIST.JN2, ACNTHIST.JN3
DB_JNLSZ = 500
```

関連dmSQL

CONNECT

START DATABASE

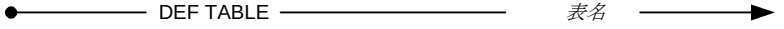
TERMINATE DATABASE

6.3 DEF TABLE

目的

特定の表のスキーマを表示します。

構文



説明

特定の表のスキーマを表示します。システム表に使用することはできません。

使用例

➲ **使用例 1a**

表を作成する:

```
dmSQL> create table tb_tmp(c00_serial serial, c01_int integer, c02_char
char(20))
```

➲ **使用例 1b**

コマンドを実行する:

```
dmSQL> “def table tb_tmp”
```

➲ **結果**

```
dmSQL> def table tb_tmp;
create table SYSADM.TB_TMP (
 C00_SERIAL  SERIAL(1),
 C01_INT  INTEGER default null ,
 C02_CHAR  CHAR(20) default null )
 in DEFTABLESPACE  lock mode row  fillfactor 100 ;
```

関連dmSQL

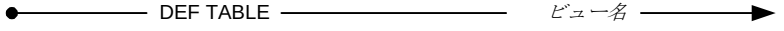
DEF VIEW

6.4 DEF VIEW

目的

ビュー定義の構造を表示します。

構文



説明

ビュー定義の構造を表示します。システム・ビューに使用することはできません。

使用例

➲ 使用例

```
dmSQL> CREATE VIEW view_tmp as select c00_serial, c01_int from tb_tmp;
dmSQL> DEF VIEW view_tmp;
dmSQL> def view view_tmp;
create view SYSADM.VIEW_TMP as select c00_serial, c01_int from SYSADM.TB_TMP
```

関連dmSQL

DEF TABLE

6.5 DISCONNECT

目的

データベース接続を切断します。

構文

説明

DISCONNECT文は、現在使用中のデータベース接続を切断します。他のデータベース接続には何の影響も与えません。CONNECT以上のセキュリティ権限をもつユーザーなら誰でも実行することができます。

AUTOCOMMITモードは、トランザクションのコミットを制御します。AUTOCOMMITモードがONのときは、各SQL文が一つのトランザクションとして取り扱われます。SQL文は、成功終了ならば自動的にコミットされ、実行エラーが発生すればロールバックされます。AUTOCOMMITモードがOFFのときは、二つのCOMMIT WORK文の間の全てのSQL文が一つのトランザクションを形成します。トランザクションで変更されたデータは、COMMIT WORKを実行するとコミットされ、ROLLBACK WORKを実行するとロールバックされます。

AUTOCOMMITモードがOFFのときにデータベース接続を切断すると、トランザクションは中止されます。トランザクションで変更されたデータはデータベースに記録されません。

マルチユーザー・データベースは、切断されても動作し続けます。他のユーザーはアクセスすることができます。UNIXで作動するシングルユーザー・データベースは、切断されると終了します。Windowsで作動するマルチ接続データベースは、最後のユーザーが切断すると終了します。

使用例

➲ **使用例**　現在使用中のデータベース接続を切断します：

```
DISCONNECT;
```

関連dmSQL

CONNECT

COMMIT WORK

ROLLBACK

START DATABASE

TERMINATE DATABASE

6.6 EXPORT(エクスポート)

Exportコマンドは、データベース表からデータの抽出と、データのテキスト・ファイルへの挿入を容易にします。 2つの設定を使用します。 Exportコマンドインターフェイスは、コマンド・オプションを指定するために使用されます。 説明ファイルは、エクスポート・ファイル・フォーマットを指定するために使用されます。

EXPORTコマンド・インターフェイス

Exportコマンド構文は、次の通りです:<data_file> これは、データを挿入するターゲット・ファイルです。 完全パスで記述する必要があります。 data_file を指定しない場合、エクスポート・ファイル名は <table_name>_out.txtになります。

TABLE エクスポートしたい表を指定してください。

[DESCRIPTION <description_file>]　これは、結果データファイルのデータフォーマットに対する説明ファイルです。 説明ファイルで、ユーザーは結果データファイルに対して規則を指定します。 詳細については、

DESCRIPTION FILE FORMAT項を参照してください。 説明ファイルが指定されていない場合、説明ファイル名は<table_name>_out.dscになります。このファイルが存在しなければ、DBMasterは所期設定の出力フォーマットを使用します。

所期設定ファイル・フォーマットは可変フォーマットです。 これは、以下を意味します。

- カラム・デリミタとしてのTAB
- 行ターミネータとしての新規行文字
- 引用符無し
- ソース表のすべてのカラムは、表と同じ順序でエクスポートされます。

[LOG <log_file>]　　このファイルは、データをアンロードしている間に発生したエラーのログを取ります。 このオプションが指定されていない場合、所期設定のログファイル名、export.logが使用されます。

[STOP_ON_ERROR]　　エラーが発生した場合、データのアンロードの停止を指定します。 このオプションが指定されていないと、エラーが発生した場合でもデータのアンロードは続行されます。

```
EXPORT
[INTO <data_file>]
TABLE [<owner_name>.]<table_name>
[DESCRIPTION <description_file>]
[LOG <log_file>]
```

[STOP_ON_ERROR]説明ファイル

アンロード結果をフォーマットするための説明ファイルのフォーマットを指定できます。 固定フォーマットと可変フォーマットの、2種類のタイプのフォーマットを使用できます。

固定フォーマット説明ファイル

固定フォーマット説明ファイルが使用されているとき、垂直に並べるエクスポート結果のそれぞれについてカラムが必要になったとします。 並べるために使用するセパレーターは、スペース文字です。

FORMAT = FIXED　　これは、固定長データファイルの説明ファイル・フォーマットを指定します。

[LOB_FORMAT= INTERNAL | EXTERNAL]　　これは、大きなオブジェクト・タイプ(blob、clob、nclobおよびその他のファイルなど)のカラムをエクスポートしているとき、外部ファイルが生成されることを指定します。 各行の大きなオブジェクト・タイプの各カラムに対して、外部ファイルが生成されます。 このオプションが指定されていない場合、データの内容がデータファイルに埋め込まれます。

外部ファイルの名前を付けるとき、次の点を心に留めることが大切です。

blobtempdir<m>\blbtmpf<n>.<tmp | txt>.

mは、ディレクトリの1からカウントされた最小の未使用数字を指定します。

例えば、blobtempdir1、blobtempdir2、blobtempdir3と名前を付けられたディレクトリがすでに存在する場合、外部ファイルを含むために新たに作成されたディレクトリはblobtempdir4になります。

nは、ディレクトリの1からカウントされた最小の未使用数字を指定します。

ファイル拡張子名がtmpまたはtxtであるかは、エクスポートされたカラムがBLOBタイプ、FILEタイプまたはCLOBタイプのどれであるかによります。 カラム・タイプがBLOBまたはFILEであれば、ファイル拡張子名はtmpになります。 それ以外の場合、カラム・タイプはtxtです。

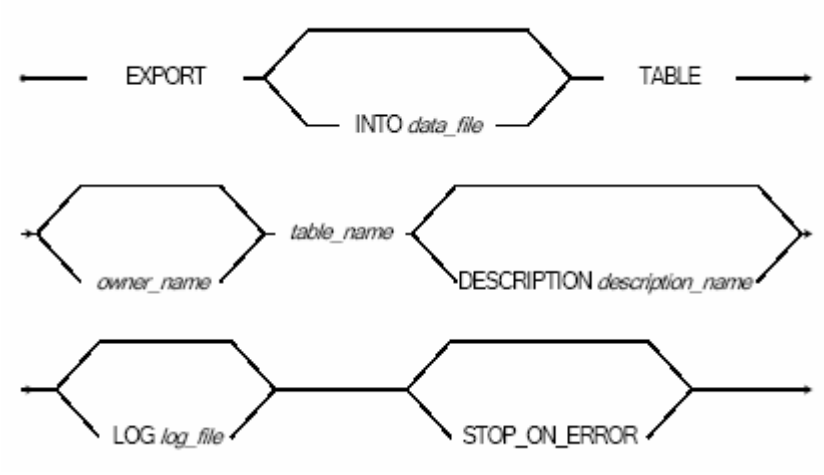
server_column_name　これは、データベースからエクスポートされるソース表カラムの名前を一覧表示します。 表名にスペースがある場合、二重引用符を使用してカラム名を囲んでください。

column_position　データファイルのカラム・バイト位置を指定します。

server_columnname とcolumn_position はスペース文字で区切られます。

column_positionは(:)で区切られる2つの数字により指定されます。例えば、1:40はデータ・ローダーがデータファイルの最初のバイトから40番目のバイトまでのデータを検索することを意味します。 データフィールドを垂直に並べるには、スペース文字を使用します。ソース表のデータがフィールド長を超える場合、データ出力は切り捨てられます。

```
FORMAT=FIXED
[LOB_FORMAT=INTERNAL | EXTERNAL]
<server_column_name> <column_position>
```



可変フォーマット説明ファイル

可変フォーマット説明ファイルを選択しているとき、結果データ出力のフィールドはユーザー指定デリミタによって区切られます。

FORMAT=VARIABLE　これは、結果出力ファイルが可変フォーマットになることを指

定します。

[COLUMN_DELIMITER=<delimiter>]　これは、データファイルの各カラムを区切る文字を指定します。 文字は、一重引用符でなければなりません。例えば、SPACEがカラム・デリミタとして使用されていることを示すには、‘ ‘を使用します。 標準文字は別として、特殊文字を表す次のエスケープ・シーケンスを例に取ります。

文字	エスケープ・シーケンス表示
TAB	\t
NEW LINE	\n

例えば、デリミタがTABの場合、<delimiter>で‘\t’を使用します。 カラム・デリミタが指定されていない場合、カラム・デリミタとしてTAB (\t)を使用します。 デリミタを選択しているときは、慎重を期してください。

カラム・デリミタの数がユーザーによって指定されたターゲット表のカラムの数より少ない場合、NULLが挿入値として使用されます。
[ROW_TERMINATOR=<row_terminator>]　この設定は行の終りを示します。
[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE]　これは、出力データが一重引用符または二重引用符で囲まれることを示します。 データに引用符がある場合、出力は2つの連続した引用符を示します。

[LOB_FORMAT=INTERNAL | EXTERNAL]: これは、大きなオブジェクト・タイプ(blob、clob、nclobおよびその他のターゲット・ファイルなど)のカラムをエクスポートしているとき、外部ファイルが生成されることを指定します。 各行の大きなオブジェクト・タイプの各カラムに対して、外部ファイルが生成されます。 このオプションが指定されていない場合、データの内容がデータファイルに埋め込まれます。

外部ファイルの名前を付けるとき、次の点を心に留めることが大切です。

blobtempdir<m>\blbtmpf<n>.<tmp | txt>.

mは、ディレクトリの1からカウントされた最小の未使用数字を指定します。

例えば、blobtempdir1、blobtempdir2、blobtempdir3と名前を付けられたディレクトリがすでに存在する場合、外部ファイルを含むために新たに作成されたディレクトリはblobtempdir4になります。

nは、ディレクトリの1からカウントされた最小の未使用数字を指定します。

ファイル拡張子名がtmpまたはtxtであるかは、エクスポートされたカラムがBLOBタイプ、FILEタイプまたはCLOBタイプのどれであるかによります。 カラム・タイプがBLOBまたはFILEであれば、ファイル拡張子名はtmpになります。 それ以外の場合、カラム・タイプはtxtです。

server_column_name: この可変は、エクスポートするサーバー表のカラムの名前を一覧表示します。 これらの名前の順序は、カラム・エクスポートの順序を表します。 そのようなリストがない場合、ソース表のすべてのカラムは表カラムと同じ順序でエクスポートされます。

```
FORMAT=VARIABLE
[COLUMN_DELIMITER=<delimiter>]
[ROW_TERMINATOR=<row_terminator>]
[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE]
[LOB_FORMAT=INTERNAL | EXTERNAL]
[<server_column_name>]
```

データ規則のインポート/エクスポート

次の表は、ファイルにデータをインポート、またはファイルからデータをエクスポートしようとしているときに適用されなければならいない規則をまとめています。

データタイプ	フォーマットのインポート/エクスポート	例
BINARY	16進数フォーマットを使用します	バイナリ数「0x004D2」をインポートするには、データファイルで004D2を使用します
CHAR	文字は排他的に使用されます	文字「inception」をインポートするには、データファイルでinceptionを使用します

NCHAR	以下の三つのフォーマットが使用できます：自動、16進形式或いはキャラクタ。 説明フラグIMPORT_NCHAR_FORMATを使用してユーザーのオプションを示します。 NCHAR_AUTOオプションはデータを16進形式としてインポートします。失敗になると、キャラクタとしてインポートします。 NCHAR_HEX オプションはデータを16進形式としてインポートします。 NCHAR_CHAR オプションはデータをキャラクタとしてインポートします。	ワード“ワード”をインポートすると、データファイルに77006f0072006400或いはワードを使用します。
VARCHAR	CHARデータタイプをご覧ください	
NVARCHAR	NCHAR データ型を参考します	
DATE	エクスポートには、フォーマットYYYY/MM/DDが使用されます	日付「2003/07/25」をインポートするには、データファイルで2003/07/25を使用します
TIME	エクスポートとインポートは、フォーマットHH:MM:SSを使用します	時間「14:30:25」をインポートするには、データファイルで14:30:25を使用します
TIMESTAMP	DATEフォーマットとDATEフォーマットの組み合わせは、TIMESTAMPのフォーマットを形成します	タイムスタンプ「2003/07/25 14:30:25」をインポートするには、データファイルで2003/07/25 14:30:25を使用します

DECIMAL	数値データ表示を使用します	数字「36.82」をインポートするには、データファイルで36.82を使用します
DOUBLE	DECIMALまたは数字の科学表記法で説明した数値データを使用します	数字「13e+12」をインポートするには、データフイルで13e+12を使用します
FLOAT	DOUBLEをご覧ください	
INTEGER	整数データを使用します	整数「576」をインポートするには、データファイルで576を使用します整数「576」をインポートするには、データファイルで576を使用します
LONG VARBINARY	2つのフォーマットを使用できます: 埋め込まれたまたは外部のファイルフォーマット埋め込まれたファイル・フォーマットの場合、16進数文字を使用します。外部ファイル・フォーマットの場合、URLが提供されます。オプションを示すには、説明フラグLOB_FORMATを使用します。 詳細については、説明ファイル仕様をご覧ください。	(1) 埋め込まれたフォーマット使用されるフォーマットはBINARYと同じです。 (2) 外部ファイル・フォーマット例えば、完全パスが“c:\MyDocument\GRAPH.GIF”であるバイナリ・ファイルをインポートする場合。提供されるURLは、c:\MyDocument\GRAPH.GIFになります

6.7 IMPORT

Importコマンドはテキストファイルからデータを抽出し、そのデータをデータベース表に挿入するために使用されます。 Importコマンドインターフェイスは、コマンド・オプションを指定するために使用されます。 説明ファイルは、インスポート・ファイル・フォーマットを指定するために使用されます。

IMPORTコマンド・インターフェイス

Importコマンド・インターフェイスは、データをインポートするためのい

くつかのオプションを提供します。 オプションには、データロード用の停止基準、エラーのログ、ソースデータファイルのデータエンコードの制御が含まれます。 ソースデータファイルののフォーマットは、説明ファイルで説明しています。

[<owner_name>.]<table_name> これは、データファイルからロードする表を識別します。 <owner_name>を指定しない場合、現在の接続ユーザーは所有者として割り当てられます。

[FROM <data_file>] これは、ロードするデータを含む実際のファイルです。data_fileを指定しない場合、データファイル名は<table_name>_in.txtになります。例えば、インポート表名がt1でデータファイル名がコマンドで指定されていない場合、データファイル名は**t1_in.txt**になります。

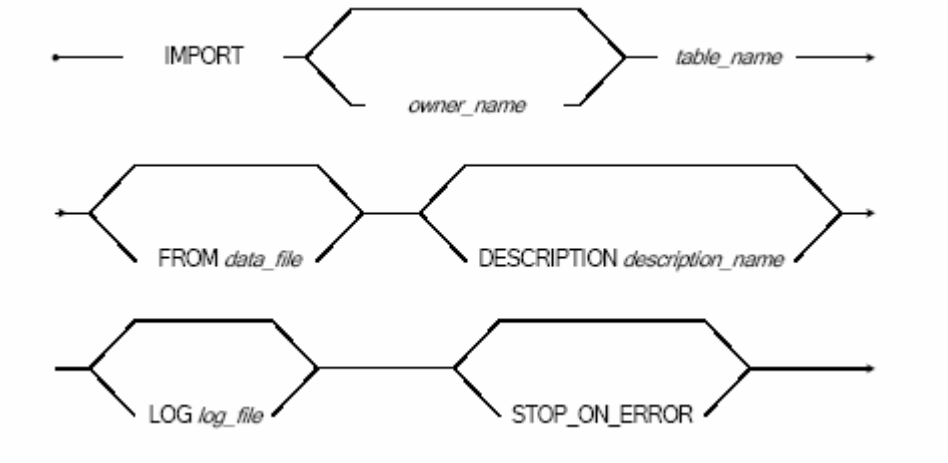
[説明 <description_file>] これは、データファイルのデータフォーマッ

トを説明する説明ファイルです。 このオプションが指定されていない場合、説明ファイル名は<table_name>_in.dscとして割り当てられます。例えば、インポート表名がt1で説明ファイルが指定されていない場合、説明ファイル名はt1_in.dscとして割り当てられます。このファイルが見つからない場合、所期設定の説明ファイル・フォーマットが可変説明ファイルフォーマットとして使用されます。

[ログ <log_file>] これは、データをロードしている間のすべてのエラーのログを取る、ログファイルを識別します。 このファイルはレコードの内容を示し、対応するエラー・メッセージだけでなくエラーを誘発します。 このオプションを指定しない場合、所期設定のログ名はimport.logになります。

[STOP_ON_ERROR] この変数が設定されている場合、インポートプロセスの間にエラーが発生すればデータのロードは停止します。 指定されていなければ、エラーが発生した時でもロードは続行されます。

```
IMPORT [<owner_name>.]<table_name>
[FROM <data_file>]
[DESCRIPTION <description_file>]
[LOG <log_file>]
[STOP_ON_ERROR]
```



説明ファイル

2種類の説明ファイルが使用されます。 1つは固定フォーマットで、もう1つは可変フォーマットです。 説明ファイルの解析エラーが明瞭に示されます。 エラー・メッセージをチェックすることによって、エラーが発生した理由を知ることができます。 エラー・メッセージは、特定の言葉を解析しているときに発生した問題を表示します。

固定フォーマット説明ファイル

FORMAT=FIXED　フォーマットが固定に設定されているとき、これは説明ふぁいるが固定長データファイル用のフォーマットを説明していることを意味します。

[START_WITH_ROW=<row_number>]　データのロードを開始するレコードを指定することができます。 このオプションを指定しない場合、所期設定の数字は1です。

START_WITH_ROWがデータファイルのデータの合計行より大きい場合、データはロードされません。 row_numberは正数でなければなりません。

[NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=<number>] これにより、それぞれのコミット・トランザクション間でロードされるレコードの行の間隔を指定します。このオプションが指定されていない場合、DBMasterは5行ごとにトランザクションをコミットします。 変数が-1に設定されている場合、コミットはされません。 この場合、効率的にロードしたければトランザクションを手動でコミットする必要があります。 変数が0に設定されている場合、インポート全体が単一のトランザクションとして表示されます。 ロードが終了した後、システムはコミットを発行します。

レコードをロードしているときにエラーが発生しても、コミットされた行数はレコードをカウントします。

例えば、NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=10を設定すると、4番目のレコードをロードしているときにエラーが発生します。 最初のレコードから3番目のレコード、および5番目のレコードから10番目のレコードがコミットされ、最初のレコードから10番目のレコードが1つのトランザクション単位として表示されます。 もちろん、STOP_ON_ERRORが指定されているとき、5番目のレコードから10番目のレコードはコミットされず、最初のレコードから3番目のレコードだけがコミットされます。

このオプションは、自動コミットがオフになっているときのみ有効です。

[LOB_FORMAT= 内部 | 外部] clob/blobフォーマットが内部であれば、データファイルのテキストはインポートされるデータとして表示されます。 さもなければ、テキストはインポートされる外部ファイルに対するURLとして表示されます。

server_column_name これは、データファイルからインスポートされるターゲット表カラムの名前を一覧表示します。 表カラム名にスペースまたはイコール記号がある場合、二重引用符を使用してそれを囲んでください。

column_position これはデータファイルのカラム・バイト位置です。

server_column_name とcolumn_position はスペース文字で区切られます。

column_positionは(:)で区切られる2つの数字により指定されます。例えば、1:40はデータ・ローダーがデータファイルの最初のバイトから40番目のバ

イトまでのデータを検索することを意味します。 データフィールドを垂直に並べるには、スペース文字を使用します。ソース表のデータがフィールド長を超える場合、行データの残りは切り捨てられます。各ラインは、ローダーがWindowsプラットフォームであるかどうかによって、新しいラインかキャリッジ・リターンと新しいラインのどちらかで終了します。 ラインが最大位置よりも短い場合、足りない部分を埋めるためにスペースが挿入されます。 ラインが最大位置より長い場合、ラインの残りは無視されます。

```
FORMAT=FIXED
[START_WITH_ROW=<row_number>]
[NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=<number>]
[LOB_FORMAT=INTERNAL | EXTERNAL]
<server_column_name> <column_position>
```

注： フィールド、server_column_name、およびcolumn_positionはスペースで区切られます。

➲ **固定フォーマット説明ファイルをインポートする例は、次の通りです:**
データファイルは次のようになります:

```
ダボリオ ナンシー 販売員 ミズ・
フラー アンドリュー 販売担当、部長 ドクター・
リーバーリング ジャネット 販売員 ミズ・
ピーコック マーガレット 販売員 ミセス・
ブキャナン スティーヴン 販売部長 ミスター・
スヤマ マイケル 販売員 ミスター・
キング ロバート 販売員 ミスター・
```

このデータファイルの説明ファイルはこのように表示されます:

```
START_WITH_ROW=1
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=5
名前 1:20
地位 20:45
性別 50:54
```

可変フォーマット説明ファイル

```
FORMAT=VARIABLE
[START_WITH_ROW=<row_number>]
[NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=<number>]
[{COLUMN_DELIMITER=<delimiter>}]
[ROW_TERMINATOR=<row_terminator>]
[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE]
[ESCAPE_CHAR=はい|いいえ]
[LOB_FORMAT=内部 | 外部]
[<server_column_name> <column_number>]
```

FORMAT=VARIABLE　これは、このファイルに可変長説明ファイル用のフォーマットが含まれていることを意味します。

[START_WITH_ROW=<row_number>]　データのロードを開始するレコードを指定することができます。 このオプションを指定しない場合、所期設定の数字は1です。START_WITH_ROWがデータファイルのデータの合計行より大きい場合、データはロードされません。 row_numberは正数でなければなりません。

[NUMBER_OF_ROWS_FOR EACH_TRANSACTION=<number>]　これにより、それぞれのコミット・トランザクション間でロードされるレコードの行の間隔を指定します。このオプションが指定されていない場合、DBMasterは5行ごとにトランザクションをコミットします。 変数が-1に設定されている場合、コミットはされません。 この場合、効率的にロ

ードしたければトランザクションを手動でコミットする必要があります。変数が0に設定されている場合、インポート全体が単一のトランザクションとして表示されます。 ロードが終了した後、システムはコミットを発行します。

レコードをロードしているときにエラーが発生しても、コミットされた行数はレコードをカウントします。

例えば、NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=10を設定すると、4番目のレコードをロードしているときにエラーが発生します。 最初のレコードから3番目のレコード、および5番目のレコードから10番目のレコードがコミットされ、最初のレコードから10番目のレコードが1つのトランザクション単位として表示されます。 もちろん、STOP_ON_ERRORが指定されているとき、5番目のレコードから10番目のレコードはコミットされず、最初のレコードから3番目のレコードだけがコミットされます。

このオプションは、自動コミットがオフになっているときのみ有効です。

[COLUMN_DELIMITER=<delimiter>] これは、データファイルの各カラムを区切る文字を指定します。 文字は、一重引用符でなければなりません。例えば、SPACEがカラム・デリミタとして使用されていることを示すには、‘ ‘を使用します。 標準文字は別として、特殊文字を表す次のエスケープ・シーケンスを例に取ります。

文字	エスケープ・シーケンス表示
TAB	¥t
NEW LINE	¥n

例えば、デリミタがTABの場合、<delimiter>で‘\t’を使用します。 カラム・デリミタが指定されていない場合、カラム・デリミタとしてTAB (\t)を使用します。 デリミタを選択しているときは、慎重を期してください。

カラム・デリミタの数がユーザーによって指定されたターゲット表のカラムの数より少ない場合、NULLが挿入値として使用されます。

[ROW_TERMINATOR=<row_terminator>] これは、行の終りを示す文字列です。.row_terminatorは、二重引用符でなければなりません。 カラム・デリミタ用のエスケープ・シーケンス規則は行のターミネータに適用されます。 それだけでなく、キャリッジ・リターンもエスケープ・シーケンスになることができます。

文字	エスケープ・シーケンス表示
キャリッジ・リターン	¥r

例えば、キャリッジ・リターンと新しいライン文字が行ターミネータを形成する場合、<row_terminator>は“\r\n”になる必要があります。 行ターミネータが指定されていない場合、新しい行文字(‘\n’)が行ターミネータとして使用されます。 行ターミネータの文字数は、2以上でなければなりません。

カラム・デリミタはrow_terminatorに入れないことにご注意ください。

[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE] これは、データソースファイルの1つのフィールドのアルファベット・データが引用符で囲まれているかどうかを示します。SINGLE_QUOTEが指定されている場合、一重引用符で囲まれたデータはデータの1つのカラムとして表示されます。DOUBLE_QUOTEが指定されている場合、二重引用符で囲まれたデータはデータの1つのカラムとして表示されます。

[ESCAPE_CHAR=YES | NO] これは、エスケープ・シーケンス(\)が使用されているか堂かを示します。 初期設定は[はい]です。 エスケープ・シーケンスが使用されている場合、エスケープ・シーケンスの後のカラム・デリミタ文字は実際のデータとして表示されます。

例えば、カラム・デリミタとしてTABの使用を指定する場合、ESCAPE_CHARが[はい]なら、\TABデータはカラム・デリミタの変わりにデータのTABとして表示されます。 行ターミネータの場合、このエスケープ文字はラインが続き、\nが実際のデータとして表示されることを意味します。 この規則は、引用符にも適用されます。

[LOB_FORMAT= 内部 | 外部] clob/blobフォーマットが内部であれば、データファイルのテキストはインポートされるデータとして表示されます。 さもなければ、テキストはインポートされる外部ファイルに対するURLとして表示されます。

server_column_name これは、データファイルからインスポートされるターゲット表カラムの名前を一覧表示します。 表カラム名にスペースまた

はイコール記号がある場合、二重引用符を使用してそれを囲んでください。

column_number　　これは、データファイルの各フィールドの基数です。

server_column_name と column_numberはスペース文字で区切られます。

注：　server_column_name と column_numberが指定されていない場合、データファイルのすべてのカラムはデータファイルカラムと同じ順序でターゲット表カラムにインポートされることにご注意ください。 つまり、データファイルの最初のカラムは表の最初のカラムとしてインポートされ、データファイルの2番目のカラムは表の2番目のカラム、等々としてインポートされます。データファイルのカラム数がターゲット表のカラム数より大きい場合、データファイルの残りのカラムは無視されます。 他方、データファイルのカラム数がターゲット表のカラム数より小さい場合、ターゲット表の残りのカラムはNULL で挿入されます。

初期設定可変フォーマット説明ファイル

ユーザーは、オプションで初期設定フォーマット用説明ファイルを指定することができます。 説明ファイルを指定しない場合、初期設定説明フォーマットが説明ファイルと見なされます。 初期設定フォーマットは、次の説明ファイル(Win32プラットフォームでは、ROW_DELIMITER=”\r\n”)が使用されることを意味します:

```
START_WITH_ROW=1
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=5
COLUMN_DELIMITER=”\t”
ROW_TERMINATOR=”\n”
```

➲ **可変フォーマット説明ファイルをインポートする例は、次の通りです:**
データファイルは、次のようになります:

```
ダボリオ ナンシー、販売員、ミズ・
フラー アンドリュー、”部長、販売担当”、ドクター・
リバーリング ジャネット、販売員、ミズ・
```

```
ピーコック マーガレット、販売員、ミセス・
ブキャナン スティーヴン、販売部長、ミスター・
スヤマ マイケル、販売員、ミスター・
キング ロバート、販売員、ミスター・
```

このデータファイルの説明ファイルはこのように表示されます:

```
START_WITH_ROW=1
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=5
COLUMN_DELIMITER=",”
ROW_TERMINATOR="\n"
DOUBLE_QUOTE
名前 1
地位 2
性別 3
```

データ規則のインポート/エクスポート

次の表は、ファイルにデータをインポート、またはファイルからデータをエクスポートしようとしているときに適用されなければならいない規則をまとめています。

データタイプ	フォーマットのインポート/エクスポート	例
BINARY	16進数フォーマットを使用します	バイナリ数「0x004D2」をインポートするには、データファイルで004D2を使用します
CHAR	文字は排他的に使用されます	文字「inception」をインポートするには、データファイルでinceptionを使用します

NCHAR	以下の三つのフォーマットが使用できます：自動、16進形式或いはキャラクタ。 説明フラグIMPORT_NCHAR_FORMATを使用してユーザーのオプションを示します。 NCHAR_AUTOオプションはデータを16進形式としてインポートします。失敗になると、キャラクタとしてインポートします。 NCHAR_HEX オプションはデータを16進形式としてインポートします。 NCHAR_CHAR オプションはデータをキャラクタとしてインポートします。	ワード“ワード”をインポートすると、データファイルに77006f0072006400或いはワードを使用します。
VARCHAR	CHARデータタイプをご覧ください	
NVARCHAR	NCHAR データタイプをご覧ください	
DATE	エクスポートには、フォーマットYYYY/MM/DDが使用されます	日付「2003/07/25」をインポートするには、データファイルで2003/07/25を使用します
TIME	エクスポートとインポートは、フォーマットHH:MM:SSを使用します	時間「14:30:25」をインポートするには、データファイルで14:30:25を使用します
TIMESTAMP	DATEフォーマットとDATEフォーマットの組み合わせは、TIMESTAMPのフォーマットを形成します	タイムスタンプ「2003/07/25 14:30:25」をインポートするには、データファイルで2003/07/25 14:30:25を使用します

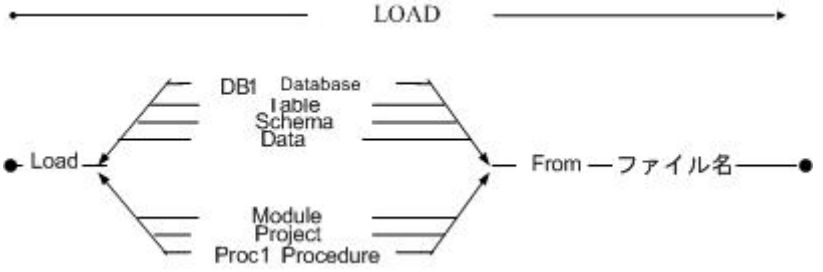
DECIMAL	数値データ表示を使用します	数字「36.82」をインポートするには、データファイルで36.82を使用します
DOUBLE	DECIMALまたは数字の科学表記 法で説明した数値データを使用します	数字「13e+12」をインポートするには、データフイルで13e+12を使用します
FLOAT	DOUBLEをご覧ください	
INTEGER	整数データを使用します	整数「576」をインポートするには、データファイルで576を使用します 整数「576」をインポートするには、データファイルで576を使用します
LONG VARBINARY	2つのフォーマットを使用できます: 埋め込まれたまたは外部のファイルフォーマット埋め込まれたファイル・フォーマットの場合、16進数文字を使用します。 外部ファイル・フォーマットの場合、URLが提供されます。 オプションを示すには、説明フラグLOB_FORMATを使用します。 詳細については、説明ファイル仕様をご覧ください。	(1) 埋め込まれたフォーマット使用されるフォーマットはBINARYと同じです。 (2) 外部ファイル・フォーマット例えば、完全パスが“c:\MyDocument\GRAPH.GIF”であるバイナリ・ファイルをインポートする場合。提供されるURLは、c:\MyDocument\GRAPH.GIFになります

6.8 LOAD

目的

テキストファイルからデータベースオブジェクトをロードします。

構文



説明

LOAD文は、テキストファイルにアンロードされているデータベースオブジェクトをデータベースに転送するdmSQLのツールです。ロードオプション：Database、Table、Schema、Data、Project、Module、Procedureがあります。同じオプションでアンロードしたファイルだけをロードすることができます。例えば、Databaseは、Databaseオプションでアンロードしたテキストファイルからロードします。

ロードするときは、自動的にトランザクションをコミットするSQL文の個数を設定します。初期設定値は1000です。この値は、ロードの速さとトランザクションが成功するかどうかに関係します。値が大きいと、ジャーナルフルでトランザクションが失敗し易くなります。値が小さいと、コミット時間が増してロードを遅くします。

ロード中にエラーが発生すると、エラーメッセージをログファイルに記録してロードを続けます。ログファイルは、実行したSQL文をundoするのにシステムが使用します。ログファイルは、外部テキストファイルと同じディレクトリに格納されます。

LOAD [DB | DATABASE]

LOAD DB文は、新規データベースに内容を転送するときに使用します。最初にデータベースを外部テキストファイルにアンロードしておき、次にLOAD DBを使用してテキストファイルからデータベースの内容をロードします。データベースは、ロードする前に作成しておきます。データベース名は、元の名前と違っていてもかまいません。DBAまたはSYSADMだけが実行することができます。

➲ **使用例　DBMaster 3.6 以降は**loaddb**オプションの設定が追加されました：**

```
Set loaddb [safe | fast]
```

safeモードは、ジャーナルモードに設定します。ロード中にエラーが発生すると、直近のコミット文までロールバックされます。エラーメッセージが画面に表示され、ログファイルに書き出されます。

fastモードは、DBMaster 3.6以前の設定です。ジャーナルを使用せずにロードします。fastモードはロードを速くしますが、エラーが発生するとデータベースを終了します。

例えば、**dmconfig.ini**ファイルに指定されていない表領域の作成がロードファイルに含まれているとします。safeモードでは、メッセージ

「ERROR(8002): [DBMaster]Configファイルにキーワードの入力が必要です」が返され、LOAD文がロールバックされます。fastモードでは、メッセージ「ERROR(30017): [DBMaster]非ジャーナルモードでエラーが発生したので、データベースを終了します」が表示されます。

注：　初期設定のオプションは “set loaddb safe” です。

LOAD TABLE

LOAD TABLE文は、表のスキーマとデータをテキストファイルからロードします。表をロードするときは、表名が重複しないことを確かめておきます。

LOAD SCHEMA

LOAD SCHEMA文は、表スキーマだけをテキストファイルからロードします。データはロードしません。スキーマをロードするときは、表名が重複しないことを確かめておきます。

LOAD DATA

LOAD DATA文は、表データだけをテキストファイルからロードします。表は、予め作成されていなければなりません。3.6以前のバージョンでは、LOAD DATAでエラーが発生すると直近のコミット文までロールバックされます。

➲ **使用例　DBMaster 3.6 以降は次のオプションの設定が追加されました：**

```
Set loaddata skip [error] | stop [on error]
```

loaddata skip error オプションは、データロード中の次のエラーメッセージをスキップします：

ERROR(401)	一意キーの違反
ERROR(410)	参照整合性制約の違反；親キーに値が無い
ERROR(6521)	表またはビューが無い
ERROR(6002)	構文エラー
ERROR(6015)	不完全に入力された *SQL* 文

エラーをスキップして、後続のロードが続けられます。上記のエラーは、データロード中に多く発生するエラーです。*loaddata stop* または *stop on error* オプションは、エラーが発生するとロード文全体をロールバックします。初期設定の設定は、*loaddata skip* [*error*] です。データロード中に発生した全てのエラーメッセージは、ログファイルに書き出されます。

LOAD MODULE

LOAD MODULE文は、モジュールをテキストファイルからロードします。

LOAD PROJECT

LOAD PROJECT文は、プロジェクトをテキストファイルからロードします。

LOAD [PROC | PROCEDURE]

LOAD PRODECURE文は、ストアド・プロシージャをテキストファイルからロードします。

使用例

- **使用例1**　ファイル “**empdb**” からデータベースをロードします。ロード中は、**100 SQL**文毎に自動的にコミットします。ファイル “**empdb**” と同じディレクトリにログファイル “**empdb.log**” が作成されます：

```
dmSQL> load db from empdb 100;
```

- **使用例2**　ファイル “**empfile**” から表をロードします。ロード中は、**50** SQL文毎に自動的にコミットします：

```
dmSQL> load table from empfile 50;
```

- **使用例3**　ファイル “**datafile**” からデータをロードします。ロード中は、初期設定の設定を使用して **1000** SQL文毎に自動的にコミットします：

```
dmSQL> load data from datafile;
```

関連dmSQL

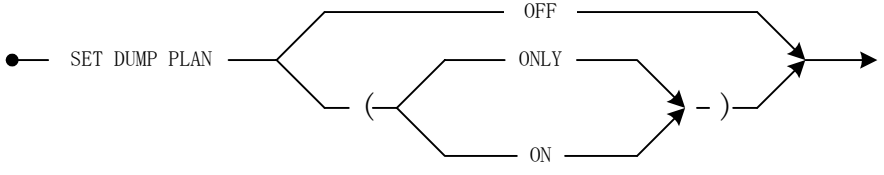
UNLOAD

6.9 SET DUMP PLAN

目的

ダンプ計画をON／OFFにします。

構文



説明

ダンプ計画は、複数のONブロックで構成されています。問合せオプティマイザは、問合せを複数の論理的なONブロックに分離し、最適化します。単純なジョイン問合せでは、通常1つのONブロックを生成します。副問合せのような複雑な問合せは、主ブロックと副ブロックから成る複数のONブロックを生成します。

オプティマイザは、各ONブロックのコストに基づき、最適な実行計画を見つけます。又、複数のPLブロックに各ONブロックを分解します。各PLブロックは、スキャンや、ジョイン等の操作を表します。

SET DUMP PLAN ONは、ダンプ計画をONにします。問合せ計画を表示し、SQL文を実行します。

SET DUMP PLAN OFFは、ダンプ計画をOFFにします。問合せ計画を表示せず、SQL文を実行します。これは初期設定です。

SET DUMP PLAN ONLYは、ダンプ計画をONにします。問合せ計画を表示しますが、SQL文は実行しません。

使用例

➲ **使用例1**

```
dmSQL> SET DUMP PLAN ON;

dmSQL> SELECT * FROM tb_tmp WHERE c01_int;



----- begin dump plan -----
{ON Block 0}

ON Type    : SCAN



[PL Block 0]

Method    : Scan
```

```
Table Name : t1

Type      : Table Scan

Order     : <none>

Factors   : (1) t1.c1 > 1

I/O Cost   : 101.0

CPU Cost   : 25.3

Sub Cost   : 0.0

Result Rows: 330.0

----- end dump plan -----

   C1          C2
=========== ===========
         2          3
1 rows selected
```

➲ 使用例2

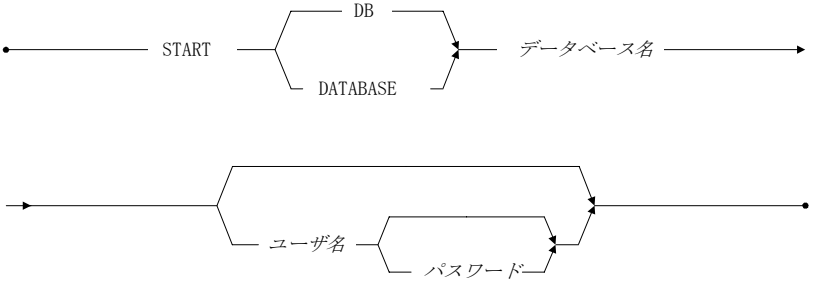
```
dmSQL> SET DUMP PLAN OFF;

dmSQL> SELECT * FROM t1 WHERE c1>1;


   C1          C2
=========== ===========
         2          3

1 rows selected
```

6.10 START DATABASE

目的

データベースをシングルユーザー・モードで起動します。

構文



データベース名	起動するデータベースの名前
ユーザー名	データベースを起動するユーザーの名前
パスワード	*ユーザー*のパスワード

説明

START DATABASE文は、データベースを他のユーザーが接続できない状態で起動します。ローカル・データベースに対してのみ使用できます。全てのユーザーが実行することができます。

ユーザー名とパスワードを指定せずにデータベースを起動させると、**dmconfig.ini**ファイルのDB_USRIDとDB_PASWDキーワードの値が使用されます。

注： **dmconfig.ini***ファイルは普通のテキストファイルなので、読み込み許可をもつ人であれば誰でもパスワードを見ることができます。このキーワードは、データベースのセキュリティ問題を起こす恐れがあります。セキュリティの必要の無いコンピュータでのみ使用してください。*

使用例

➲ **使用例**　ユーザー**vivian**が**Employees**データベースを起動します：

```
START DATABASE Employees vivian shuka828;
```

関連dmSQL

CONNECT

CREATE DATABASE

DISCONNECT

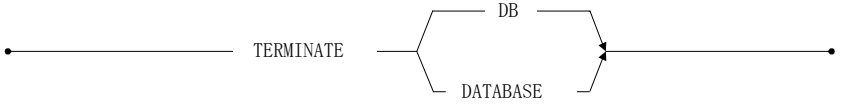
TERMINATE DATABASE

6.11 TERMINATE DATABASE

目的

データベースを終了します。

構文



説明

TERMINATE DATABASE文は、データベースを終了し接続できなくします。通常、クライアント／サーバー・データベースに対して使用します。DBAまたはSYSADMだけが実行することができます。

使用例

➲ **使用例**　起動しているデータベースを終了します：

```
TERMINATE DATABASE;
```

関連dmSQL

CONNECT

CREATE DATABASE

DISCONNECT

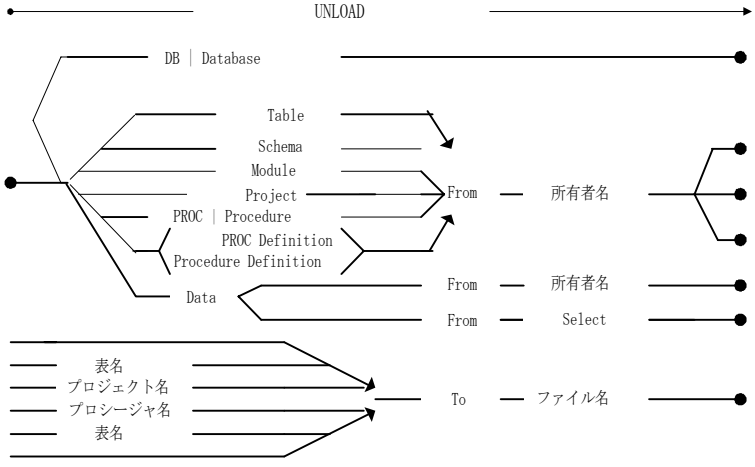
START DATABASE

6.12 UNLOAD

目的

データベースオブジェクトをテキストファイルにアンロードします。

構文



説明

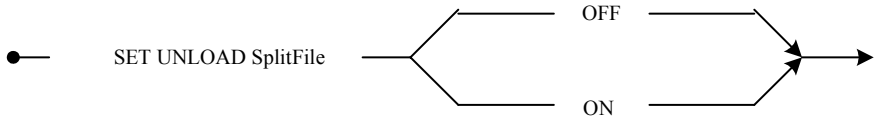
UNLOAD文は、dmSQLでデータベースの内容を外部テキストファイルに転送するツールです。アンロードが成功すると、データベースオブジェクトを設定するスクリプトファイル(拡張子 s0)と、BLOBデータファイル(拡張子 bn)が作成されます。

UNLOADのオプションにはdatabase、table、schema、data、project、module、procedure、procedure definitionがあります。SELECT権限があるオブジェクトだけをアンロードすることができます。例えば、ある表だけにSELECT権限をもっているとすると、その表だけアンロードすることができます。データベース全体をアンロードすることができるのは、DBAまたはSYSADMだけです。

表の定義、データ、索引と他の情報に則して**set unload splitfile on/off**を使用してアンロードスクリプトファイルを分かれます

Set unload splitfile on ……アンロードスクリプトファイルを分かれます

Set unload splitfle off ……<external text file name>.bn anと<external text file name>.soファイルのみ自動にアンロードされる。.



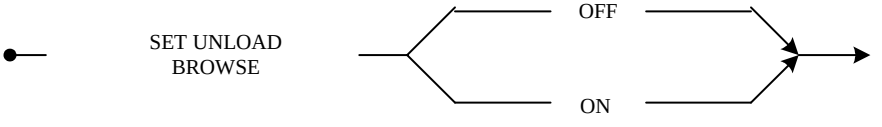
➲ 使用例

```
dmSQL> set unload splitfile on;
dmSQL> unload db to empdb;
dmSQL> set unload splitfile off;
```

ユーザーは **set unload browse on/off** コマンドを使用してアンロードコマンドとほかのDMLが同時に使用できること、アンロード操作をする前アンロードされたデータの一致性を確保します。 ディフォルトはOFFです。

Set unload browse on ……アンロードコマンドとほかのDMLが同時に使用できます。つまり、ダーティデータはアンロードされます。

Set unload browse off ……アンロードコマンドとほかのDMLが同時に使用できません。



➲ 使用例

```
dmSQL> set unload browse on;
dmSQL> unload db to empdb;
dmSQL> set unload browse off;
```

UNLOAD [DB | DATABASE]

DBAまたはSYSADMは、データベース全体を外部テキストファイルにアンロードすることができます。ファイルには、セキュリティ、 表領域、 定義、索引、シノニム、データ等に関する情報がアンロードされます。各データベースに、少なくとも二つの外部ファイル：スクリプトファイルとBLOBデータファイルが生成されます。

外部テキストファイル名は**empdb**です。初期設定では、ファイルは現在の作業ディレクトリに作成されます。この例では、少なくとも二つのテキストファイル**empdb.s0**と**empdb.b0**が作成されます。BLOBファイル**empdb.b0**がオペレーティング・システムの上限サイズを超えるときは、最大99までの**empdb.b1**、**empdb.b2**、 …、**empdb.bn**が順に生成されます。スクリプトファイルは1個だけ生成されます。**emodb.s0**の最大サイズは、オペレーティング・システムの上限値です。

UNLOAD TABLE

表をアンロードします。表データだけでなく、定義、シノニム、索引、主キー、外部キーも記録されます。

所有者名と表名には、ワイルドカード“_”と“%”を使用することができます。“_”は任意の一文字を表し、“%”は任意の文字列を表します。この二つはDOSの “?” と “*” と同じ働きをします。

UNLOAD SCHEMA

このオプションはUNLOAD TABLEと非常に良く似ていますが、表の定義だけをアンロードし、表データはアンロードしません。UNLOAD TABLEと同じワイルドカードを使用することができます。

UNLOAD DATA

このオプションは表データをアンロードします。表定義はアンロードされません。前の二つのオプションと同じワイルドカードを使用することができます。アンロードする表のSELECT権限をもつユーザーだけが実行することができます。

DBMaster 3.6以降のバージョンでは、別のUNLOAD DATA構文： **unload data from (select文) to ファイル名**もサポートされます。ただし、DELETE、INSERT、UPDATEのDDL文は使えません。SELECTカラムは、同じ表のものでなければなりません。

➲ **使用例1** 正しい構文：

```
dmSQL> UNLOAD DATA FROM (SELECT tb_doc.c01_int , tb_doc.c02_cha FROM tb_doc,
tb_txt where tb_doc.c01_int= tb_txt.c01_int) to f1;
```

➲ **使用例2** 不正な構文：

```
dmSQL> UNLOAD DATA FROM (SELECT tb_doc.c01_int, tb_txt.c01_int from tb_doc,
tb_txt where tb_doc.c01_int = tb_txt.c01_int) to f1;
```

集計関数や組み込み関数をカラムに使用することはできません。

➲ **使用例3** 不正な構文：

```
dmSQL> UNLOAD DATA FROM (SELECT avg(c01_int) FROM tb_doc) TO f1;

dmSQL> UNLOAD DATA FROM (SELECT now() FROM tb_doc) TO f1;
```

ビューやシノニムを使用することはできます。

➲ **使用例4** 正しい構文：

```
dmSQL> UNLOAD DATA FROM (SELECT * FROM syn_tmp WHERE c01_int > 10) TO f1;

dmSQL> UNLOAD DATA FROM (SELECT * FROM view_tmp WHERE c01_int < 10) TO f1;
```

UNLOAD PROJECT

プロジェクトをアンロードします。

UNLOAD MODULE

モジュールをアンロードします。

UNLOAD [PROC | PROCEDURE]

ストアド・プロシージャをアンロードします。

UNLOAD [PROC DEFINITION | PROCEDURE DEFINITION]

ストアド・プロシージャの定義をアンロードします。

使用例

➲ **使用例1**　現ユーザーの“**e tab**”表をアンロードします。表名の中に空白があるときは二重引用符で囲います：

```
dmSQL> UNLOAD TABLE FROM “e tab” TO empfile;
```

➲ **使用例2**　**SYSADM**所有の**emptab**、**empname**、… 等 **emp**で始まる表をアンロードします：

```
dmSQL> UNLOAD TABLE FROM SYSADM.emp% TO empfile;
```

➲ **使用例3**　全ての所有者の**ktab**表の定義をアンロードします：

```
dmSQL> UNLOAD SCHEMA FROM %.ktab TO kfile;
```

表名の中にワイルドカード文字が含まれているときは、エスケープ文字“\” を使用するか、または二重引用符で囲います

➲ **使用例4**　表名**abc%**のデータをアンロードします：

```
dmSQL> UNLOAD DATA FROM abc\% TO abcfile;
dmSQL> UNLOAD DATA FROM “abc%” TO abcfile;
```

関連dmSQL

LOAD